MASTER IN COMPUTER SCIENCE

A prototype for lessons learned systems oriented towards safety-critical software

Defat, Simon; Jeunejean, Félix

Award date:
2004

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année Académique 2003 - 2004

A prototype for lessons learned systems oriented towards safety-critical software

Simon Defat & Félix Jeunejean

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique.

Résumé

Selon le rapport réalisé par le groupe de travail de l'*"Association for Computing Machinery"* sur le *"Licensing of Software Engineers Working on Safety-Critical Software"* [Knight & al., 2001], un système permettant de rassembler les accidents survenus lors des développements logiciels de systèmes critiques devrait être mis en place, afin d'étudier les causes de ces accidents logiciels et d'y remédier. Une analyse ainsi qu'une structuration et une présentation appropriées de ces données pourraient avoir un impact important sur la prévention d'accidents futurs similaires.

Nous proposons d'employer une approche basée sur le knowledge management - un *"lessons learned system"* implanté sur le web - afin d'atteindre cet objectif. Les objectifs du *"lessons learned system"* proposé sont de capturer et de fournir des leçons qui pourront bénéficier à d'autres utilisateurs, opérateurs et ingénieurs, dans le but d'augmenter la connaissance qui peut mener à l'établissement de produits et dispositifs logiciels plus sûrs. Afin de valider cette approche, nous avons élaboré un prototype exploratoire "jetable".

Abstract

According to the influential report by the *"Association for Computing Machinery Task Force"* on *"Licensing of Software Engineers Working on Safety-Critical Software"* [Knight & al., 2001], a reporting system to collect and evaluate data on what undermines the software development of safety-critical systems should be established; on the other hand, a proper analysis, which consists of the structuring and presentation of these data, could have a substantial impact on the prevention of future accidents.

We propose to use a knowledge management approach - a web-based lessons learned system - to help achieve this purpose. The goals of the proposed lessons learned system are to capture, as well as to provide lessons that can benefit users, operators and engineers by increasing the knowledge, which can lead to safer software-related products and devices. In order to validate this approach, we built an exploratory throwaway prototype.

Acknowledgements

Before starting the writing of our master thesis, we would like to thank our supervisor Andres Silva, teacher at the Polytechnical University of Madrid, and of course our promoter Patrick Heymans, as well as Pierre-Yves Schobbens, professors at the University of Namur, who allowed us to carry out our training course in Madrid. We express our gratitude for their patience and judicious advice, as well as for the interest they expressed throughout the realization of this project.

Our acknowledgements also go to Paloma Jimenez and César Hernández Martín, our roommates during our stay in Madrid, and all our Spanish friends for their welcome. We would like to personally express that, in our virtual world, we are lucky to have quite real parents!

Finally, we insist on bringing in our work all the people who have allowed us, by their advice, their assistance, their encouragements and their availability, to bring this project to a successful conclusion.

To all of them, we say "thank you".

Contents

List of Figures

Introduction

Context

This master thesis has been written in the context of our master degree in computer science at the University of Namur. Its subject is tightly linked with our training course at the Polytechnical University of Madrid (UPM). Our thesis comes within the scope of the works of [Silva & al., 2002], lecturer at the UPM.

Subject

Accident analysis and reporting systems are very important for the safety of many organizations in the world. The goal of this master thesis is to explain the principles of Lessons Learned (LL) systems that allow exchanges of information between software engineers and computer scientists. It concerns software accident investigation, as well as reports in a variety of application fields, such as aviation, aerospace, biomedical industries, military systems, or nuclear applications.

According to the influential report by the *"Association for Computing Machinery (ACM) Task Force"* on *"Licensing of Software Engineers Working on Safety-Critical Software"* [Knight & al., 2001], a reporting system to collect and evaluate data on what went wrong in the software development of safety-critical systems should be established, and a proper analysis, structuring and presentation of these data could have a substantial impact on the prevention of future accidents.

We propose to use a Knowledge Management (KM) approach - a web-based lessons learned system - to help achieve this purpose. The goals of the proposed LL system are to capture and provide lessons that can benefit users, operators and engineers by increasing the knowledge that can lead to safer software-related products and devices.

Content

First part: state of the art

We start by describing the state of the art. We then present the problems with current methods and approaches about accident reporting and software. The second chapter explains the main principles and concepts of KM and shows how an approach based on KM can be used in order to build a LL system oriented towards safety-critical software. The last chapter is about LL systems; it focuses mainly on the LL process which consists of five tasks: collection, verification, storage, dissemination and reuse. It gives definitions of LL and explains the different roles, requirements and types of LL systems.

Second part: analysis and implementation

We proceed with the analysis and implementation of the conceived prototype. First, we describe the typology and the various attributes that compose a LL oriented towards safety-critical software. We then explain the development method we used, which is mainly inspired from the spiral model (definition of features, class diagram and use cases, design of the database, implementation, tests). We finish by illustrating some screenshots of the application we conceived, which consists of an exploratory throwaway prototype.

Third part: creation of an environment for lessons learned through knowledge management

We demonstrate that the success of LL systems, and more generally KM systems, heavily depends on the culture of the organization. Cultural barriers must be necessarily taken into account. Some recommendations to incite and improve sharing knowledge, as well as using LL systems are also given in this third part.

Fourth part: ethical questions

Ethical questions about the functioning and the use of LL systems are exposed in the fourth part.

Fifth part: future works

Finally, we give ideas that could improve the efficiency of LL systems. We approach inter alia the possible interaction between different systems, the benefits of conceiving a generic LL system, as well as the concept of monitored distribution.

Sources

This master thesis is mainly based on the unpublished article by [Silva & al., 2002]. The following papers have also helped us during our writing: [van Heijst & al., 1997], [Rus & al., 2002], [GAO, 2002] and [Weber & al., 2001].

Part I

State of the art

Chapter 1

Accident reporting and software

1.1 Introduction

Accident investigation and reporting systems play a primary role in the safety of many industries across the globe. Currently, almost all complex systems and safety-related devices depend on software to perform their tasks. This concerns a high number of crucial fields such as aviation, aerospace, automotive industry, chemical industry, healthcare, military systems, marine systems, rail industry, or nuclear power. These fields' safety-related functions depend on software and therefore on software engineering. Nowadays both practitioners and scientists widely recognize that software engineering is still unperfected, regardless of fifty years of progress [Jackson, 2001]. Current methods are not always efficient. This advice is supported by the fact that several software-related accidents have occurred in the past and, unfortunately, others are expected to happen in the future [Peterson, 1996], [Neumann, 1995], [Wiener, 1993].

What we should essentially realize is that every accident, failure or mishap can help everyone, as it is an opportunity to learn to avoid similar catastrophes in the future. Collecting, analyzing and publishing accidents can be very positive exercises, and learning from failed experiences is more constructive than learning from successes [Petroski, 1994]. *"Human beings have assessed negative experiences since ancient times, as an indication of what they should not attempt to do. The cleverest human beings learned their lessons from the failures of others, but most people learned from their own, even repetitive, errors"* [Minsky, 1996].

1.2 Examples of current accident reporting systems

This section illustrates some problems, mainly technical, concerning current reporting systems. These problems should be taken into account before developing and deploying such systems.

1.2.1 Comp.risks forum

One of the most important media on the web used to collect and distribute information about software-related accidents is the comp.risks forum. However, some engineers have noted that this approach is inefficient when it comes to extract general lessons and retain the most important information about an accident. For example, several accident reports are not completed, and some lessons are reported by writers who are not experts in the field of the accident they describe [Silva & al., 2002].

1.2.2 After Action Review program

Another example is the *"After Action Review"* program of the Center for Army Lessons Learned (CALL), that increases the performance of the US Army. According to this program, every successful or failed mission should be followed by an inquiry about what actually happened, what was expected to happen, the difference between both, and, finally, what can be learned from it. A specific application of this program takes place each time a disaster has occurred. In this situation, the concerned authorities and the affected organizations perform an analysis and write a report. The problem is that these reports seldom evolve into real lessons designed for general and practical reuse [Silva & al., 2002].

1.3 Identified problems of current systems

According to the influential report by the ACM [Knight & al., 2001], *"an anonymous reporting system to collect and evaluate data on what went wrong in software development of safety-critical systems should be established, and a proper analysis, structuring and presentation of these data could have a major impact on the prevention of future accidents. This is already being done in other fields (FAA/NASA Air Safety Reporting System, Marine Safety Reporting System...), but not in software engineering"* [Knight & al., 2001].

Several recent studies and workshops [Fisher & al., 1998], [GAO, 2002], [Johnson & al., 2000], [Reimer, 1998] [Secchi & al., 1999], [Weber & al., 2001]

established that:

- Systems to collect and disseminate crucial information are underused. Most of them are managed by various governments and organizations. For example, the survey carried out by the NASA [GAO, 2002] asserts that there is no guarantee that information stored in its knowledge system will be reused in future missions. This survey reveals weaknesses in the collection and sharing of crucial information which is not routinely identified, collected, or shared by programmers and project managers. Employees are dissatisfied with the system. They do not have enough time to share knowledge and do not trust the system. On the other hand, they have also the impression that using the system is not advantageous.

- There was strong evidence of the distribution process' weaknesses and few organizations performed a costs/benefits analysis on the impact of their KM systems.

- None of the observed organizations implemented a process that actively and intelligently distributes reported accidents to interested users.

- Generally, accident and error reporting systems poorly satisfy their initial goal of encouraging knowledge reuse and sharing. This problem is related to the textual representation of the knowledge assets as a set of free-text fields (lack of structure), and also to the bad incorporation of software systems into the processes, which they are intended to support. Impractical representation and integration with internal processes are therefore the main reasons of current accident reporting methods. Consequently, one could generally point out that the main problem is that accident reports about software engineering are imprecise and difficult to understand, and that the most important information is very hard to find among these reports.

1.4 Solution based on KM and LL systems

The problem also comes from the fact that sometimes knowledge is fuzzy, unclear and uncertain. It is therefore natural that a solution to the aforementioned problems may come from KM. LL systems or repository based LL systems are build from KM, which allows to extract a specific piece of knowledge at a specific place and time. Such systems support the main tasks of developing, combining, distributing and consolidating knowledge [van Heijst & al., 1997]. For example, the solution found by a worker in Montreal who faces a particular software problem can be submitted by him

and reused by another worker in Tokyo. Accordingly, why not build and use LL systems for managing lessons related to safety-critical software systems? Such systems should provide [Silva & al., 2002]:

- A way to collect, store and distribute LL from/to the right people.
- Values for all the attributes that characterize a LL.
- A solution to index LL, in order to support their collection and future dissemination.
- A solution to allow users to define the LL they are interested in.
- A solution to maintain the accuracy, consistency and integrity of the LL repository.

1.5 Conclusion

There is a clear need to gather LL from accidents in the field of safety-critical software and to distribute them among the interested parties. The same mistakes and errors are repeated all the time. This is why it is possible and highly desirable to build a system that can prevent it from happening in the future.

Chapter 2

Knowledge management in software engineering

2.1 Introduction

The previous chapter focused on the problems of current reporting techniques related to software accidents. The current one demonstrates the utility of using KM based on a LL system. We first define KM and then describe the processes it supports. Afterwards, we explain the forms of learning present in an organization. We will proceed with the definition of needs and roles of KM in software engineering. The chapter ends with an explanation of the various types of KM artifacts.

2.2 KM definition

The concept of KM emerged in the mid-1980s from the need to sort the high amount of information that organizations need to manipulate. In the 1990s, many industries started to use the term KM in association with commercial computer technologies [Rus & al., 2002].

The aim of KM is to capitalize on the intellectual property of an organization, as well as to increase productivity, cooperation and innovation in the workplace. The purpose also is to shorten development times, reduce costs and risk, as well as to increase performance, quality and scientific return [GAO, 2002].

Holtshouse [Powers, 1999], Corporate Strategy Director for Xerox, divided KM into ten distinct areas:

- *Sharing knowledge and best practices.*
- *Instilling responsibility for knowledge sharing.*

- *Capturing and reusing past experiences.*
- *Embedding knowledge in products, services, and processes.*
- *Producing knowledge as a product.*
- *Driving knowledge generation for innovation.*
- *Mapping networks of experts.*
- *Building and mining customer knowledge bases.*
- *Understanding and measuring the value of knowledge.*
- *Leveraging intellectual assets.*

2.3 Individual and organizational levels of learning

"Capturing and reusing past experiences" is the most relevant domain with regard to LL systems. Activities related to KM involve learning, capturing and reusing experience. Learning is a fundamental part of KM. Learning experience is a process that occurs at two interacting levels, the individual and the organizational (or group) levels.

- **Knowledge spreads from groups to individuals**, who have to assimilate shared knowledge before they can use it for a specific task. This implies that learning is a basic mainstay.
- **Knowledge also spreads from individuals to groups**. *"KM aims to elevate individual knowledge to the organizational level by capturing and sharing individual knowledge and turning it into knowledge that the organization can access"* [Rus & al., 2002]. As Senge says, *"Organizations only learn through individuals who learn. Individual learning does not guarantee organizational learning. But, without it, no organizational learning occurs"* [Senge, 1994].

2.4 Explicit and tacit knowledge

We can distinguish two forms of knowledge: explicit knowledge and tacit knowledge. **Explicit knowledge**, or codified knowledge, is expressed knowledge and is generally easy to use. It corresponds to information and skills that someone can easily communicate and document, as processes, templates and data. **Tacit knowledge** is personal knowledge that employees accumulate by experience. This knowledge can be very difficult to express and is often subjective [Rus & al., 2002].

Knowledge is also characterized by its scope, which indicates when and where it is applicable, to whom it is accessible, and which activities it supports [Rus & al., 2002].

2.5 KM systems and supported processes

A **KM system**, also called a **knowledge repository**, is a tool which supports KM. [van Heijst & al., 1997] distinguishes four basic processes: development, consolidation, distribution, and combination.

- **Developing/creating new knowledge**: Members of organizations acquire knowledge through learning, problem solving, work in R&D departments, innovation, failure analysis, daily experiences, creativity, and acquirement from other sources. KM systems can support this first process by recording knowledge.
- **Consolidating new and existing knowledge**: By storing knowledge in a repository, it becomes persistent over time and it can be retrieved easily. Consequently, knowledge is accessible at the right time and place, and can be delivered to individuals who need it.
- **Distributing knowledge**: Knowledge should be actively or passively distributed to those who need it. KM systems need features to decide who should be informed about this or that knowledge.
- **Combining available knowledge**: An organization can increase its performance if its available knowledge areas are combined in new products. KM systems make the access to knowledge developed in other departments of the organization easier.

A KM system should be organized in order to support each of these processes, but taking each of them individually is not enough. All processes interact in complex ways. In order to be efficient, organizations should take this interaction into account.

2.6 Forms of learning

According to [van Heijst & al., 1997], there are two main types of learning in an organization: top-down learning and bottom-up learning.

Top-down learning means that someone at a high or management level estimates that a particular knowledge domain will be promising, and he decides to do all the necessary actions to obtain it.

*"**Bottom-up learning** refers to the process where a worker, either at the management level or on the work floor, learns something which might be useful and then this **lesson learned** is distributed through the organization. The term LL represents any positive or negative experience or insight that can be used to improve the performance or the security of the organization in the future"* [van Heijst & al., 1997].

Bottom-up learning has three sub-types: **individual learning**, **learning through communication**, and **learning with a knowledge repository**. An organization should develop and promote each form of learning, because it is an intellectual capital that cannot be lost. The three sub-types are complementary and occur in parallel.

2.6.1 Individual learning

Employees accumulate personal experience through their daily work and use it to enhance the work processes of their organization. They create new knowledge. In fact, this personal experience may be a kind of practical LL for the organization. However, individual learning is not always and everywhere possible. In order to allow employees learning from themselves, they need to get reactions and opinions about the accuracy and the pertinence of their LL, and their consequences on the work processes of the organization. All workers also need to have some freedom in deciding how they do their jobs, otherwise they cannot be able to experiment.

With individual learning, the problem is that knowledge is not collected and disseminated for reuse. This is the reason why a **LL system** needs to be elaborated, allowing workers to distribute and share their personal experiences. Obviously, this need depends on the **size** and the **nature** of the organization.

Figure 2.1 shows the process of individual learning.

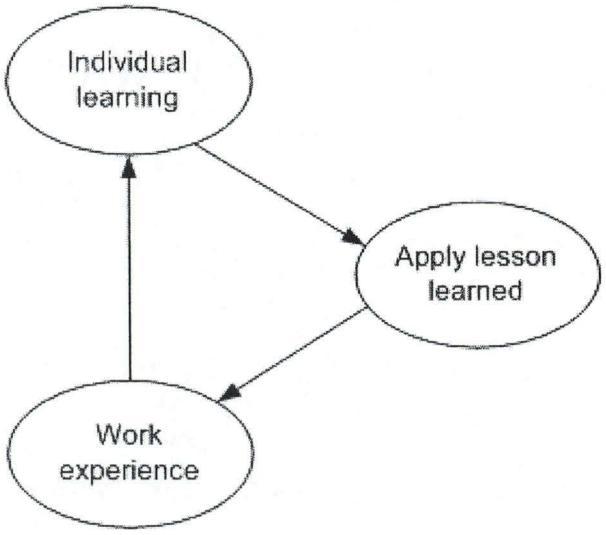

Figure 2.1: Individual learning [van Heijst & al., 1997].

2.6.2 Learning through communication

Learning through communication allows employees to share their personal experience and knowledge among them. This starts necessarily from individual learning. Learning through communication is obviously complementary with individual learning and is more efficient, because LL are shared throughout the organization.

Organizations should encourage their employees to share their experience throughout this communication process, and should develop an environment in which it is **rewarding** for them to share positive and negative experiences. There are different ways of communicating knowledge depending on how many workers have to be informed: a few people (personal casting), a complete department or organization (broadcasting), or only directly concerned and interested people (narrow casting).

Learning through communication completes and reinforces individual learning because knowledge is created, shared or combined with other knowledge. However, if we consider that knowledge is volatile, a repository could be helpful, as it will allow the retaining of knowledge developed from individual learning and communication.

Figure 2.2 shows the process of learning through communication.

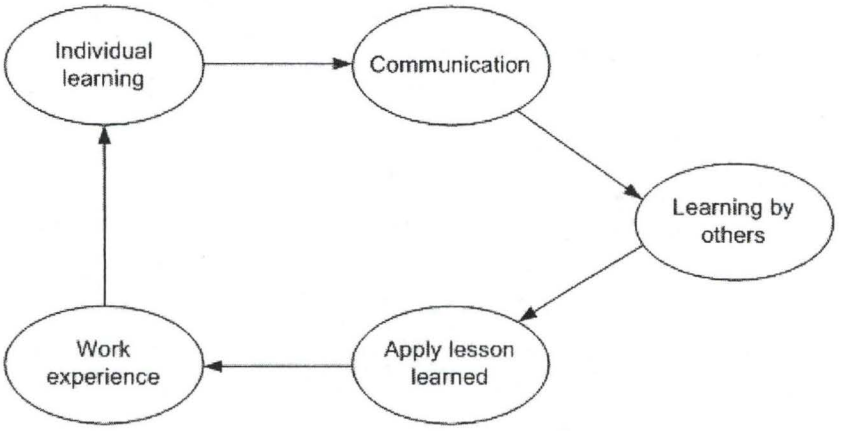

Figure 2.2: Learning through communication [van Heijst & al., 1997].

2.6.3 Learning with a knowledge repository

The knowledge repository will be used in order to store LL developed by individual learning and learning through communication. It is the same as learning through communication but communicating pieces of knowledge is replaced by collecting, verifying, storing, disseminating and reusing these pieces.

The main role of the organization in this kind of knowledge sharing is to motivate and encourage employees to **take time** to write and submit their LL into the knowledge repository, a sometimes difficult task.

Figure 2.3 shows the process of learning with a knowledge repository, while figure 2.4 shows how the forms of learning interact with the processes supported by KM.

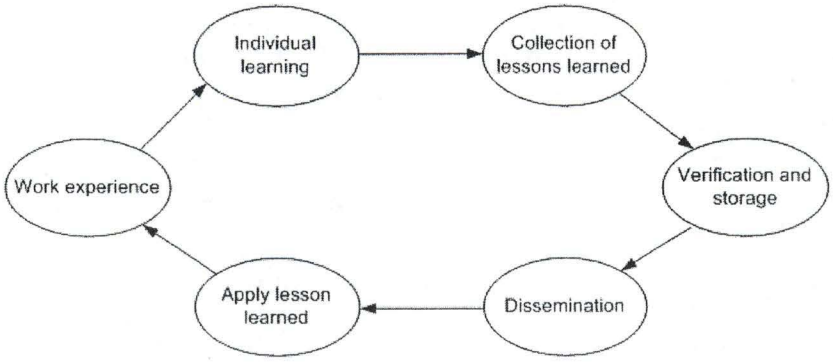

Figure 2.3: Learning with a knowledge repository [van Heijst & al., 1997].

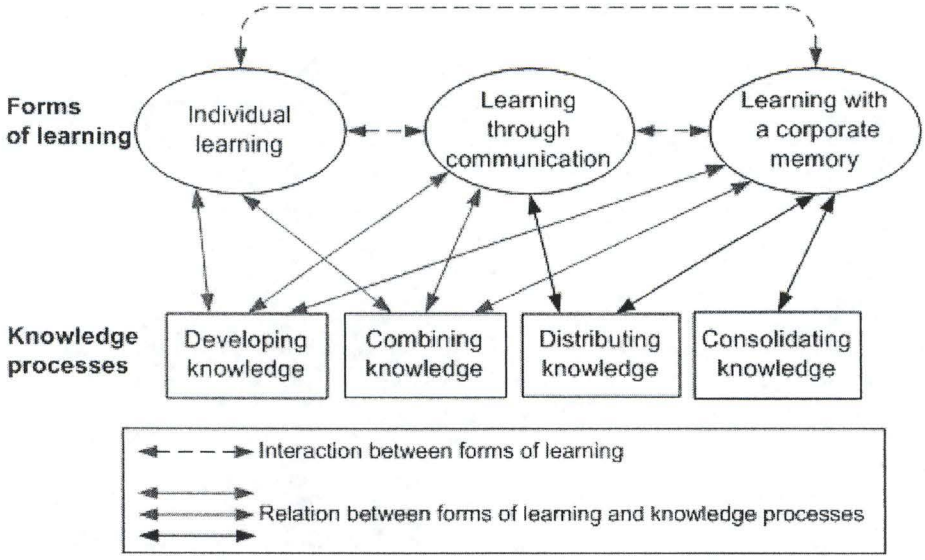

Figure 2.4: Interaction between forms of learning and relation with knowledge processes [van Heijst & al., 1997].

2.7 Needs of KM in software engineering

The main asset of a software organization is its intellectual capital. Knowledge in software engineering is diverse, enormous and regularly increasing. Hence, organizations have problems in identifying the content, the location and the use of knowledge. The basic motivation and driver for KM in software engineering is an improved use of its primary asset: knowledge [Rus & al., 2002].

2.7.1 Business needs

Decreasing time and cost and increasing quality

Organizations need to decrease the development time and cost of software projects. Reducing errors reduces rework. If the context is similar, repeating successful processes increases productivity and favors future successes. For this reason, organizations should apply the knowledge acquired in previous and similar projects. Unfortunately, developers do not reuse past experience frequently and repeat well known errors within organizations. Software engineers gain experience in each project and both organizations and individuals learn more if they share it [Rus & al., 2002].

Making better decisions

In software development, each employee is involved in the decision process. Most of the time, decisions are based on individual knowledge and informal content. In a small organization, this way of proceeding is often workable, but in a large organization, inefficient. The individual knowledge should be shared and leveraged at project and organization level, in such a way that employees can take better decisions across the organization. Some knowledge is also shared by informal exchange. Experienced developers can share their experience with inexperienced developers in an informal way. Nonetheless, informal capturing and sharing is not enough. The only way for everyone to access the needed knowledge is if organizations formalize ways of knowledge sharing [Rus & al., 2002].

2.7.2 Knowledge needs

Knowing who is knowledgeable is necessary for organizations to easily access the good persons, to efficiently staff project, to identify training needs, to match employees with training offers and to create a strategy to prevent valuable knowledge from disappearing. If individuals own information that is not explicitly captured, organizations can leverage that knowledge only if they manage to identify and access these people [Rus & al., 2002].

The knowledge of software organizations is related to various areas. Organizations have to acquire knowledge about new technologies and master them, as, most of the time, project members resort to the *"learning by doing"* approach, which costs a lot of time. Software development also requires an access to knowledge about the application domains for which software is being developed [Rus & al., 2002].

2.8 Roles of KM in software engineering

[Rus & al., 2002] distinguishes three roles of KM in software engineering: supporting core software engineering activities, supporting product and project memory, and supporting learning and improvement.

2.8.1 Supporting core software engineering activities

The first role consists of supporting core software engineering activities. It includes:

- **Document management** focuses on authoring, reviewing, editing and using documents, which become the organization's assets in capturing explicit knowledge. Document management systems allow employees to share documented knowledge.
- **Competence management and expert identification** is a solution to the problem of tracking experts who own important undocumented knowledge. Competence management systems (or skills management systems) analyze email repositories, as well as documents, and build keyword-based profiles that characterize each employee. Afterwards, organizations can use them to identify experts in a specific domain.
- **Software reuse** intends to reduce programmers' rework. Programmers continuously reinvent the wheel. In order to prevent it, software reuse suggests to record software which would be useful for others in a repository.

2.8.2 Supporting product and project memory

The second role of KM in software engineering consists of supporting product and project memory. Learning from practice requires a product and project memory. Memory is built through version control, change management, documenting design decisions, and requirements' traceability. Therefore, each version of a document has information about who, when and why the change was made.

2.8.3 Supporting learning and improvement

The last role of KM in software engineering consists of supporting learning and improvement. Most of the time, the project managers' personal experience guide their decisions; the problem is that not all of them have enough experience. Building, using and improving predictive models, which can **guide decision making** for future projects based on past projects, become a natural part of KM strategy.

2.9 KM artifacts

The field of KM does not only cover LL but also best practices, incident reports and alerts [Silva & al., 2002]:

- **Best practices** refer to examples and cases that illustrate the good use of something. They capture only successful stories.
- **Alerts** aim to warn an organization about particular problems with a particular technology.
- **Incident reports** describe negative experiences or accidents and explain them.

As we will see in the next chapter, these artifacts are too restrictive to be considered as LL.

Chapter 3

Lessons learned systems

3.1 Introduction

The first chapter explained the problems that lead to the setup of a LL system oriented towards safety-critical software. The second chapter highlighted the need to build a system based on KM.

This chapter provides definitions of LL; it explains the goals of LL systems oriented towards safety-critical software and describes the LL process, as well as the different types of LL systems resulting from it. It ends with an explanation of the general qualities and requirements of LL systems.

LL systems are based on KM. They have been settled in commercial, government, and military organizations since the early 80's to collect, store, distribute and share knowledge and experience. Nevertheless, their ability to encourage knowledge and experience sharing was limited [Weber & al., 2001]. Current LL systems are not often used and the main proof of it is that many of them contain large amounts of non-reused information. It is therefore crucial to exactly understand what a LL system should be, which are its features and which processes they are intended to support.

3.2 LL definitions

3.2.1 LL artifact

There are several definitions of LL in formally written resources. We have chosen the last definition of the list below, because we think that it is the most adequate and complete. Moreover, this definition explains clearly the reuse of a LL.

- *"LL were originally conceived as guidelines, tips, or checklists of what went right or wrong in a particular event"* [Stewart, 1997]. Concerning organizations who want to improve their performance with a LL system, this definition is out of date, because they have now adopted criteria to check, accept and validate lessons for rightness in their systems.

- *"A LL is the change resulting from applying a lesson that significantly improves a targeted process"* [Bartlett, 1999]. This definition expresses one of the main goals of LL systems.

- *"A LL is a good work practice or innovation approach that is captured and shared to promote repeated application. A LL may also be an adverse work practice or experience that is captured and shared to avoid recurrence"* [DOE-STD-7501-99, 1999].

- *"A LL is the knowledge acquired from an innovation or an adverse experience that causes a worker or an organization to improve a process or activity to work safer, more efficiently, or with higher quality"* [Bickford, 2000].

- The United States Air Force gives a more concrete definition: *"A LL system is a recorded experience of value; a conclusion drawn from analysis of feedback information on past and/or current programs, policies, systems and processes. Lessons may show successes or innovative techniques, or they may show deficiencies or problems to be avoided"* [Weber & al., 2001] . A lesson may be:

  1. An informal policy or procedure.
  2. Something you want to repeat.
  3. A solution to a problem, or a corrective action.
  4. How to avoid repeating an error.
  5. Something you never want to do (again).

- *"A LL is a knowledge or understanding gained by experience. The experience may be **positive**, as in a successful test or mission, or **negative**, as in a mishap or failure. Successes are also considered sources*

> *of LL. A lesson must be* ***significant*** *in that it has real or assumed impact on operations;* ***valid*** *in that it is factually and technically correct; and* ***applicable*** *in that it identifies a specific design, process, or decision that reduces or eliminates the potential for failures and mishaps, or reinforces a positive result"* [Secchi & al., 1999].
>
> This definition is the most complete and is currently used by the American, European, and Japanese Space Agencies. It clarifies the criteria needed to reuse lessons and how it should focus on processes that a lesson can impact [Weber & al., 2001].

Furthermore, [Silva & al., 2002] underlines the fact that *"the goals of a LL system are to capture and provide lessons that can benefit users, operators, engineers and, society, by increasing the knowledge that can lead to safer software-related products and device"*. He considers that a LL is a *"knowledge artifact derived from a* ***negative experience*** *that suggests the means to avoid the occurrence of similar negative experiences in the future"*. This definition only focuses on negative experiences, unlike the definition of Secchi. We will consider the definition of Secchi but, as Silva, restrict our work to negative experiences, due to the prohibitive costs of errors and failures in safety-critical software field. Positive experience is useful, and could be incorporated into a LL system. However, negative examples are much more useful in safety because the main focus of the engineer is to avoid negative experiences (accidents). Therefore, after an accident took place, the analysis of its causes leads to better measures to avoid same accidents in the future. It also is difficult in safety-critical field to define exactly what a positive experience is and what we can really learn from that to avoid other accidents in the future.

3.2.2 Other KM artifacts

In section 2.9, KM artifacts (best practices, incident reports and alerts) have been defined. Under the light of the definition we just adopted, we can say that these such KM artifacts are not considered as LL mainly because a LL must be significant, valid, applicable and must derive from a negative experience.

- **Best practices** capture only successful (positive) stories. They are not necessarily derived from specific experiences.

- **Alerts** are derived from negative experiences and are aimed to warn an industry of particular problems with a particular technology. They are just "alarms", i.e. that there is no necessarily an accident which has happened (yet).

- **Incident reports** describe accidents but never give a solution, suggestion or recommendation to solve the problems caused by theses accidents (negative experiences).

Best practices, incident reports and alerts suffer from inconsistency, imprecision and incompleteness [Silva & al., 2002].

	Originates from experiences	Describes failures	Describes successes	Describes accidents	Describes solutions
LL	Yes	Yes	No	Yes	Yes
Incident report	Yes	Yes	No	Yes	No
Alert	Yes	Yes	No	No	No
Best practice	Possibly	No	Yes	No	Yes (1)

Table 3.1: Differences between KM artifacts.

(1) Best practices do not really contain solutions because they are not related to accidents (problems) or not necessarily derived from a specific experience.

If an alert happens, we have a failure but not yet an accident (the cause and the solution of the failure are not identified). If an accident occurs afterwards, this alert *may be transformed* into an incident report (the cause and the solution of the accident are not still identified). Once the cause and the solution of the accident are identified, the incident report *may become* a LL.

3.3 Goals of LL systems oriented towards safety-critical software

This section explains the main goals and aims of a LL system oriented towards safety-critical software, the identity of interested organizations or industries and, finally, the uncertainty bound over such a system.

3.3.1 Goals

"The goal of a LL system is to increment the learning capacity of the organization in order to increase its competitiveness by continuously adapting to a changing environment" [van Heijst & al., 1997], and also *"to shorten development times, reduce cost and risk and increase performance, quality and scientific return"* [GAO, 2002]. The above goal is unsatisfying and must be adapted to the characteristics of a LL system oriented towards safety-critical software.

The first goal of this kind of system, which is also called a LL repository for safety-critical software, and the main difference with other current systems, is **not to increase the competitiveness** of an organization, but **to increase the safety** of the systems being built. It aims to harvest and deliver lessons that could help operators, engineers and, more generally, societies and communities, by augmenting the knowledge that can conduct to safer software-related products and devices.

A LL oriented towards safety-critical software is a piece of knowledge constructed from a negative experience that gives solutions in order to prevent the recurrence of similar experiences in the future.

3.3.2 Collaboration and communities

We can observe that many organizations or industries collaborate in the field of safety-critical software. *"On the one hand, increasing safety is a different goal, and on the other hand, there is* ***a tendency within any industrial sector to collaborate in relation to safety issues****, as negative incidents can affect the whole sector. This tendency to cooperate is so clear that even defense-related safety procedures were transferred from the US to the USSR during the cold war, when competition between the two superpowers was at its height"* [Leveson, 1995].

Specific LL systems related to safety-critical software are mainly designed for **groups of organizations** or **particular industrial sectors** (nuclear, defense, aerospace...), and **communities**, particularly the software engi-

neering community. These systems do not target a single or particular company, because safety and security related to software is a subject that goes beyond the limits of a single organization. In these systems, communities are very important, which is the reason why the forum comp.risks exists. *"There is a similar distinction for safety and reliability standards. Most are industry-specific standards, but there are non-industry-specific standards as well. A LL can be of interest only to a particular organization or industry, but it could have* ***non-industry-specific interest****. LL can be beneficial if they are reused by other organizations that are in the same industry or share the same interests"* [Weber & al., 2001].

Currently, there are mainly LL systems for construction industries, military and government organizations. In the field of safety-critical software, similar and adequate systems for the **software engineering community** do not exist, except the comp.risks forum [Silva & al., 2002]. On the other hand, similar LL systems exist in specific organizations, such as the system established by the NASA [GAO, 2002].

3.3.3 Uncertainty

"There is some information on LL approaches for the construction industry, focused on risk management at the supply chain. But software has very different characteristics, and this leads to problems. One such problem is related to the fact that there is not always a complete or satisfactory amount of information regarding a software-related accident, nor is it always clear what the best approach for avoiding a similar accident in the future is. However, this uncertainty is not so much of a problem for a LL system, as long as it is explicitly reported, so potential receivers of this information are aware of its volatility and/or lack of justification" [Tah & al., 2001].

3.4 LL process

Subsection 2.6.3 pointed out that a knowledge repository can store the LL developed by individual learning and learning through communication. According to [Weber & al., 2001], sharing and communicating knowledge among workers is achieved by the following tasks: collecting, verifying, storing, disseminating and reusing knowledge. These five tasks represent the **LL process** which is schematized in figure 3.1.

3.4.1 Collection

[Silva & al., 2002] distinguishes two main types of collection sub-processes: the active and the passive collection.

- In the **active collection**, [Knight & al., 2000] says that *"LL are searched and scanned by humans or automatically by the system itself throughout the organization, analyzing the documents and the communications for example".*

- In the **passive collection**, workers directly submit and record their lessons into the system, using a web form for example, with specific attributes and free text-fields describing the structure of the lessons. This kind of collection takes place in 2/3 of the organizations dealing with KM processes [Weber & al., 2001].

3.4.2 Verification

The LL repository should be evaluated and validated for relevance, correctness, non-redundancy and consistency. **Experts** are needed in order to perform this task and their roles go beyond those of the moderator as in forums like comp.risks. Experts have to take decision and accept, modify or reject the LL submitted. In some cases, a dialog between them and the submitter of a LL should be necessary.

Experts will also determine whether or not *"a lesson is relevant across many other projects, is unique to a particular department or project, or applies globally to the organization as a whole"* [GAO, 2002].

3.4.3 Storage

Storage concerns the physical representation, indexation and structure of the lessons in the repository (database). It transforms lessons into persistent lessons. *"Lesson representation can be structured, semi-structured, or in different media"* [Weber & al., 2001].

3.4.4 Dissemination

There are two main types of dissemination sub-processes: the active and the passive dissemination [Weber & al., 2001], [Silva & al., 2002].

- In the **active dissemination**, users automatically receive, and generally without personal intervention, the lessons they are interested in. We can distinguish four types of active dissemination:
  - **Broadcasting**: Everybody in the organization receives the LL. Broadcasting may be sometimes useful in case of emergency, but a large number of users can receive lessons in which they are not interested.
  - **Active dissemination with user profiles**: Users have to fill in a profile to receive lessons. This allows them to be notified only of the lessons they are interested in, which results from a matching between the user profiles and the content of the LL in the repository.
  - **Proactive dissemination**: *"The system builds a model of the user's interface events to predict when to prompt users with appropriate lessons"* [Weber & al., 2001]. In other words, the LL system is incorporated into the process/application it intends to support, by listening and observing the operations performed by the user. It is a kind of monitoring, which will be presented in the last part about future works. Proactive dissemination, also called **monitored distribution**, is very difficult to build, and it is hard not to drown the users with lots of irrelevant information.
  - **Reactive dissemination with user profiles**: Users can call the system directly (via a link) to retrieve appropriate lessons. It is useful when they need to obtain additional information concerning the job they are currently doing.
- In the **passive dissemination**, users retrieve themselves lessons from the system. It is the most common form of dissemination. Users can perform different kinds of searches in the repository: hierarchical search, search by attributes (matching with the attributes of a lesson) or search by keywords. However, passive dissemination is also the most inefficient dissemination method. Recent studies and workshops have established that these kinds of systems are underused. The main reasons is that users do not necessarily know about the existence of the LL system, nor know how to use it, where to find it, and how to understand its results [Secchi & al., 1999], [GAO, 2002], [Johnson & al., 2000].

3.4.5 Reuse

In order to be reused, a LL must contain a **recommendation** or **solution**. The choice of whether to reuse a lesson's solution or recommendation is made by the user. In order to improve reuse, users should be encouraged to frequently post negative or positive **feedbacks** about the reuse of the solution posted by someone else. This will be very useful for future potential users.

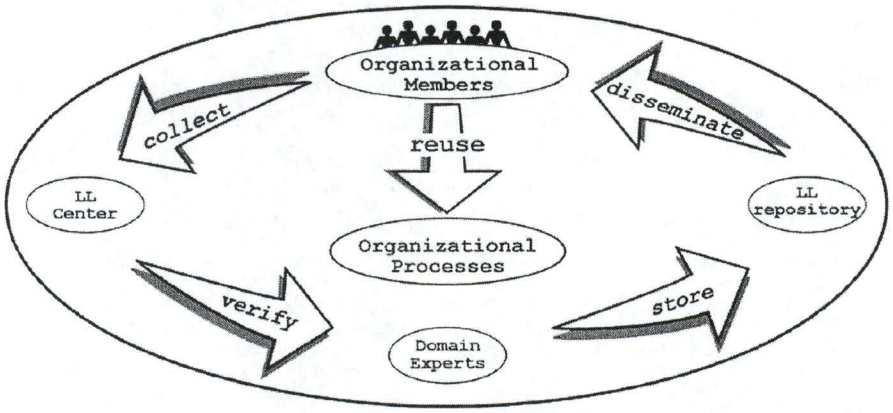

Figure 3.1: Generic LL process [Weber & al., 2001].

3.4.6 LL process and processes supported by KM

As mentioned in section 2.5, a good KM system should fill all the following roles:

- Developing new knowledge.
- Consolidating new and existing knowledge.
- Distributing knowledge.
- Combining knowledge.

Table 3.2 shows the relationships between the processes supported by KM and the LL process. Since a LL system is a kind of KM system, it is no surprise to observe that the two processes are very similar. We can see that the LL process is more precise than the KM process which does not take into account the verification of a knowledge piece. The LL process specializes the KM process which is more generic.

Processes supported by KM	**LL process**
Developing new knowledge	Collection
Consolidating new and existing knowledge	Storage
/	Verification
Distributing knowledge	Dissemination
Combining knowledge	Reuse

Table 3.2: Correspondence between processes supported by KM and LL process.

3.5 Classification of LL systems

There are different kinds of LL systems. They can be catalogued according to the collection and dissemination sub-processes, as well as according to other characteristics.

3.5.1 Collection and dissemination sub-processes

In the previous section, the LL process has been described (collection, verification, storage, dissemination and reuse). [van Heijst & al., 1997] classifies LL systems according to the collection and dissemination sub-processes. The authors distinguish different ways of doing both the sub-processes, with different implications for the users and the organizations or communities that will use the system. Table 3.3 is a classification of LL systems which depends on whether the collection sub-process is active or passive and whether the dissemination sub-process is active or passive:

- **Knowledge attic**: Collection and dissemination are both passive, generally performed through a web form. It is the simplest type of LL system to build. The best example of this kind of system is the **comp.risks forum**. It is also used in organizations like NASA. The benefit is that it is not intrusive for the employees, but the disadvantage is that knowledge attic systems are underused, because it requires a high discipline and an organizational culture (described in the chapter 12) that incites workers to use it. However, this approach seems to be suitable for a community of interests.

- **Knowledge sponge**: Collection is active and dissemination is passive. If the collection is performed by the system automatically, lessons are recorded in real time and are immediately available. This type of system is very rare; there are only but a few examples in the world [Silva & al., 2002].

- **Knowledge publisher**: Collection is passive and dissemination is active (by filling a user profile or by subscribing to a mailing list). It is used for example by the US Department of Energy. Lessons are posted by the users, which means that they need to be checked. Validation and consolidation by experts takes time and the team of experts decides to disseminate the lessons to the potential users.

- **Knowledge pump**: Collection and dissemination are both active. It is used, for example, by the US Army. Knowledge pump is the most difficult and complex type of LL system to achieve.

	Active collection	Passive collection
Active dissemination	Knowledge pump	Knowledge publisher
Passive dissemination	Knowledge sponge	Knowledge attic

Table 3.3: Types of LL systems [Silva & al., 2002].

3.5.2 Other characteristics

[Weber & al., 2001] proposes to classify LL systems according to the following characteristics:

- **Content**: Based on the content of a LL system, it is possible to make distinction between pure and hybrid systems:
  - **Pure** LL systems only include LL.
  - **Hybrid** systems also include other KM artifacts, described in subsection 3.2.2, such as alerts, incidents reports and/or best practices. For example, the hybrid system of the National Security Agency (NSA) contains three different types of LL in his system which are informational, successful or problematic.

- **Nature**: LL systems can be classified according to the nature of the processes and users they are designed to support:
  - **Planning lessons** teach something related to plan execution and their content aims to change an evolving plan in order to help achieve its goals. Typical examples are planning processes in military organizations and operations.
  - **Technical lessons** are the result of a technician's experience and refer to problems, their causes and their solutions. Technical work is not delivered through plans, but through jobs or projects. These lessons are related to technical processes which often require applying domain-specific expertise for analysis and troubleshooting.

- **Orientation:** Typically, LL systems are implemented in order to support a **specific organization**, and they should be built in accordance with that organization's goals. On the other hand, some LL systems are built to support a **group of organizations** or **community**, as explained in subsection 3.3.2 about collaboration and communities.

- **Duration**: LL systems can be **permanent** or **temporary** (due to a temporary job or event). This characteristic can depend on the **organization type**. Some organizations are **adaptable** in which case

they can quickly incorporate LL in their processes. Others are **rigid** in which case they use doctrine that is slowly updated. Adaptable organizations do not necessarily need to maintain a permanent lesson repository because lessons, once incorporated into these organization's processes, have already been learned/reused. In contrast, rigid organizations (e.g. military organizations) have a greater need to maintain lesson repositories because there is often a long time prior to the incorporation of lesson knowledge into doctrine, or lessons may not be deemed sufficiently general for inclusion into doctrine.

- **Architecture**: *"LL systems can be **standalone** or **integrated** in a targeted and internal process. Integration allows active dissemination, they can also be accessed by a link in the decision support tool"* [Bickford, 2000]. Proactive dissemination and reactive dissemination with user profiles allows a LL system to be integrated in a targeted process/application.

- **Attributes and format**: Most LL repositories have both textual and non-textual attributes. For example, a LL may be described by a set of attributes and supplemented with a video, a diagram or a report.

- **Confidentiality**: Some LL systems give access rights to the users, allowing or disallowing them to see the lessons. This implies that some lessons are **classified and restricted** (by industrial sector for example: nuclear, aerospace...), and others **unclassified**.

- **Size**: It concerns the number of LL that can be stored in the repository. Technically, it is possible to store a huge number of LL in a repository. The system will be more useful if it stores a huge number of LL. Examples about the size of LL systems are given [Weber & al., 2001]:

  - Idaho National Engineering and Environmental Laboratory (< 100 LL)
  - Department of Energy (DOE) Corporate LL Collections (100 - 1000 LL)
  - Marine Corps LL System (1000 - 5000 LL)
  - Center for Army Lessons Learned (CALL) (5000 - 10000 LL)
  - Eureka (Xerox) (> 30000 LL)

From most of the sets of values of the characteristics mentioned above, it is strongly recommended to build LL systems by choosing only one value for each of them, otherwise it complicates the future collection, storage and dissemination of lessons.

3.6 General qualities and requirements of LL systems

[van Heijst & al., 1997] listed general qualities and requirements that a LL system should satisfy.

1. **Accessibility**: Users should have facilities to access knowledge in the LL repository and to link it with other knowledge.
2. **Localization**: Users should know which other users or groups of users could have the knowledge needed for a particular activity.
3. **Profiles of interest**: Expert users should be able to choose which other users or groups of users should be interested in a particular LL. According to the active dissemination sub-process, the system should be able to recognize LL in which a user is interested.
4. **Ease of use**: Users should have a certain ease of use when submitting and retrieving a LL in the system, and it should be gratifying for them to use the system.
5. **Verifiability**: LL should be precisely defined, formulated and exactly described by well-defined attributes.
6. **Consistency**: LL repository should keep relevant, correct, no redundant and consistent.
7. **Dissemination**: LL repository should have facilities to distribute existing and new LL to the users who are interested.
8. **Reusability**: LL should absolutely include a solution/recommendation which helps to prevent occurrences of similar accidents, failures or more generally unwanted situations in the future.

Part II

Analysis and implementation

Chapter 4

Mission statement and development method

4.1 Mission statement

Before explaining in details the development method, it is important to note that the analysis presented in this part is an analysis for an **exploratory throwaway prototyping** and not for a complete application. The first goal imposed was to rapidly implement an overview of the main functionalities of a LL system oriented towards safety-critical software, to evaluate whether future research and additional work could be useful in this field.

As far as we know, there is currently no such LL systems oriented towards safety-critical software. The comp.risks forum concerns safety-critical software but this forum is inefficient and in disorder. A moderator monitors the forum but there is no structured information nor verification performed by experts. It often reports problems and accidents but do not give any solution. For example, several accident reports are not completed, and some lessons are reported by writers who are not experts in the field of the accident they describe [Silva & al., 2002].

This implies that our LL system is complementary with comp.risks but does not completely replace it. For example, it could be interesting to discuss a problem in comp.risks before to submit it in the LL system. It would allow to warn interested people earlier, while information is not yet complete and structured.

4.2 Development method

The features and the analysis described below only take into account the requirements that it was possible to implement during a work period of **four months**. Therefore, the list of features presented is not meant to be complete. In the last part of our thesis, some ideas about additional features and future applications are presented. These latter requirements will also have to be taken into account to conceive a full fledged LL system oriented towards safety-critical software.

By analyzing a few LL systems on the Internet, such as the NASA Public LL System[1], and several articles ([van Heijst & al., 1997], [Weber & al., 2001], [Silva & al., 2002]...), the main attributes of a LL system oriented towards safety-critical software have been defined. This **list of attributes** is described in the next chapter. It logically follows the state of the art established before. Afterwards, a development method has been discussed and put into operation. It consisted of the following steps:

- **Features** of the LL system: According to the LL process described in the state of the art, the features have been defined. A **level of priority** and an **estimated technical risk** have been attached to each feature, defining which features are more important than others, and should be implemented at first. The **links and dependencies** among the various features have been also defined. It is important to note that our features did not evolve during the different iterations. We used the tool *Omni-Vista OnYourMark Pro* to write our features.

- According to the features' priority, system analysis and implementation were carried out following an **iterative approach**, known as the **"spiral model"** in project management. Figure 4.1 shows this iterative approach. Each step of the development method has been reviewed, refined and validated iteration by iteration:

  - **Class diagram** and **class dictionary**: According to the list of attributes, a class diagram representing the main classes of the future application has been drafted to have an overview of the application domain. A class dictionary also describes exactly each class, its main components, and its relations with other classes. We used *UML Studio 5.0* to draw our class diagram.
  - **Use cases**: Based on the features and their priority, the various use cases of the system have been identified and linked to the appropriate feature. Use cases are based on the typology of [Cockburn, 2000]. For each use case with a user goal level, normal

[1] http://llis.nasa.gov/llis/plls/index.html

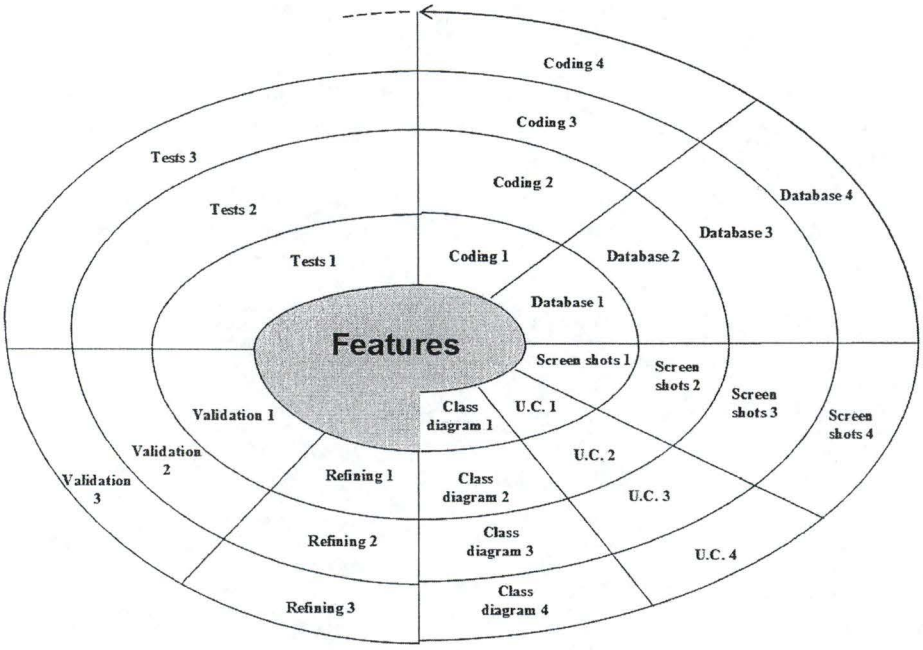

Figure 4.1: Iterative approach.

flow and alternative flows are presented. We used *Microsoft Visio* to draw our use cases.

- **Screenshots**: According to the features, the class diagram and the use cases, some basic screenshots of the future prototype have been drawn, aside from the database design. The goals of the screenshots are to have an overview of the future look of the application, the content and the sequence of the various screens, as well as the interaction with the user. Screenshots also partly allow to check whether features and use cases are fulfilled by the implementation. We used *Macromedia Dreamweaver MX* to draw our screenshots quickly.
- **Logical and physical design of the database**: A complete analysis has been carried out to design the database. We used *DB-MAIN* to draw our E/R schema. *PostgreSQL* was our database server and we used *pgAdmin III* to manage our database.
- **Coding**: It concerns the technical implementation of the prototype. The prototype has been mainly written in *PHP* and our application server was *Apache*. We used *Easy PHP* to configure and run our application, and *PHP Expert Editor* in order to write our code.
- **Tests**: Tests have been made to validate the prototype and to

check the fulfillment of the features and the use cases.

Each step of the method described in the next chapters only concerns the **last iteration**. It would have been boring to present details of each iteration.

Table 4.1 gives more details about the work performed for each iteration.

	Duration	Number of use cases (25)
Iteration 1	5 weeks	5 UC
Iteration 2	7 weeks	11 UC
Iteration 3	4 weeks	5 UC
Iteration 4	2 weeks	4 UC

Table 4.1: Iterations details.

Chapter 10 will describe the physical architecture of the system, based on a web architecture. It also explains the different tools used to build the prototype, and why we have opted for them.

Chapter 11 shows the main implementation results of the prototype and their correspondence with the appropriate use cases.

4.3 Classification of our LL system

Table 4.2 shows the characteristics of our prototype according to the classification established in section 3.5.

Collection & dissemination	Knowledge attic & publisher
Content	Pure
Nature	Technical lessons
Orientation	Group of organizations or community
Duration	Permanent
Architecture	Standalone Integrated (Reactive dissemination with macro)
Attributes & format	Textual and non-textual attributes
Confidentiality	No rights management
Size	Small (prototype)

Table 4.2: Classification of our LL system.

Chapter 5

Attributes of lessons learned systems

5.1 Attributes of general LL systems

Following the article of [van Heijst & al., 1997], [Silva & al., 2002] defines a set of general attributes identifying a LL, extracted in part from existing LL systems. This homogeneous representation is necessary because several authors employ different lesson representation in the literature. It is therefore indispensable to find a single, structured, stereotypical and effective lesson representation. Attributes are typical of any kind of LL system and they consider the goals of LL systems, the LL process and the LL requirements presented before. In the next section, attributes below are fitted to LL systems oriented towards safety-critical software and we give for each of them additional information, such as domain of values, multiplicity or default value.

- **Name**: Name, label, general identifier of the LL [1].
- **Author's identity**: Most LL systems indicate the identity of the worker who posts a LL, but some do not disclose it. The reason is that, after the author's identity is revealed, it can be used for goals other than sharing knowledge, such as job evaluation. On the contrary, it can be gratifying and sometimes rewarding for a worker to show his abilities to find and solve problems. Therefore, in some systems, it can be practical to leave to the user the option of revealing his identity.
- **Domain**: Domain to which the LL is related. An inventory of the main knowledge domains is needed.

[1] Name is a secondary identifier but there is also a technical primary identifier. We do not indicate all technical attributes in this section because it is not very useful.

- **Business processes**: Processes, organizational activities in which the LL is used. Possible values of this attribute change for every organization.

- **Organizational roles**: Roles to which the LL is attached. *"LL systems differ according to the nature of the processes (roles) and users they are designed to support. For example,military personnel execute planning processes (i.e. tasks are part of plans with established goals, usually in a multi-person and distributed context). In contrast, technicians are users whose technical processes often require applying domain-specific expertise for diagnosis and troubleshooting"* [Weber & al., 2001].

- **Relevant sources**: Sources (persons, books, software...) possessing some knowledge related to the LL.

- **Nature**: Characteristics of the LL in terms of quality and accuracy (heuristic, formal, complete, under development...)

- **Proficiency level**: Level of proficiency at which the LL is available to the organization. For example, if the LL is written for a specific technician, proficiency level will be high. On the contrary, proficiency level will be low if the lesson can be understood by everyone.

- **Stability**: Rate of change of the LL's content.

- **Time**: Date at which the LL is made available.

- **Form**: Physical/symbolic representation of the LL (video, report, manual, chart...). It could be an attachment downloadable, directly reusable by a user who consults a lesson.

- **Related knowledge assets**: Cross-references to related LL stored in the repository.

- **Related products and services**: References to other products and services that bear any relation to the LL. These products and services can be external or internal to the organization. This attribute suggests that the submitted lesson could have impact on these products and services.

- **Justification**: Method used to check the LL (proof, empirical, witness confirmation...).

- **History**: Change history, backup management of the different versions of the LL, from its origin to the current version.

- **Solution/recommendation**: In most current LL systems, another attribute is used to indicate what should be done with a particular lesson. When these conditions are met, this solution should be applied. Concerning LL related to successes, the solution consists of repeating, under similar conditions, the originating actions (occurred actions that give birth to the lesson), to be sure that the lesson contribution will cause the same results. Concerning LL related to failures or accidents, the solution consists of means that allow to avoid the repetition of the originating actions and prevent similar accidents in the future.

It has been observed that several LL systems include other attributes. For example, some contain an attribute related to **estimated saving/costs avoidance** which could help to control the efficiency of the system. Safety and cost are always in balance but, when the life of people is concerned, safety gets the better of safety. This attribute is not really suitable in the case of LL related to safety-critical software, because the goal is to increase general safety and not to make more profit or less expenses. Indeed, there are safety levels that are no negotiable. These levels must be achieved whatever the price because they concern the life of people. Additionally, it is not easy to correctly compute these estimations. However, this kind of estimations could be written by some users who submit feedbacks about a solution they have reused.

The attributes presented above are not completely suited to our LL system oriented towards safety-critical software. Some should be redefined (e.g. nature), completed (e.g. organizational role, business processes), or removed (e.g. time), and others may be missing. For example, it is also essential to ask users to complete a user profile which allows them to define they areas of interest and describe their own functions. Such profiles may be used in the active dissemination method of the LL process, by matching the lessons contained in the repository with profiles, allowing users to automatically receive lessons they want. These profiles contain several attributes (domain, business process, related products/services...) listed above but not all, otherwise user profiles and matching with lessons would be too complex.

These reasons lead us to extend and refine these attributes for the particular case of LL related to safety-critical software.

5.2 Attributes of LL systems oriented towards safety-critical software

The general attributes mentioned above can be used in any LL system. In this section, some of these attributes will be redefined and others will be added. All the attributes will represent a LL oriented towards safety-critical software.

It is important to repeat that a LL oriented towards safety-critical software reports only negative experiences (failures and accidents), and not successes. The system will only contain technical lessons and not planning lessons, because it is software related. The main goal is to improve the general safety and not to increase the competitiveness. This system is aimed at the software engineering community. These characteristics and the next attributes presented distinguish our LL system from all other systems.

In summary, a LL:

- contains **general attributes** which characterize and index it,
- is linked to an **accident** which gives birth to the lesson, and which is described by one or many **accident event sequences**,
- has one or many **solutions** on which users can post **feedbacks** or **evaluations**,
- is posted by one user and sent to others, using **user profiles**,
- is checked and validated by an **expert**.

It is important to note that several attributes are filled by the system **automatically** (see default value) and others are **optional** (see multiplicity). This is to make the encoding of LL as fast and easy as possible to the user. As we will see later, this is the role of the experts to check and determine most optional attributes.

5.2.1 LL general attributes

- **Name**
  - *Definition*: Name, label, general identifier of the LL [2].
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: Why software bug causes shuttle countdown hold at T-31 seconds.
  - *Source*: [Silva & al., 2002]
- **Author's identity**
  - *Definition*: Users have the choice to reveal their identity or not. This attribute is included in the user profile described in subsection 5.2.5.
  - *Domain of values*: Undefined string (existing user)
  - *Multiplicity*: 0-1
  - *Default value*: Yes
  - *Example*: Karl Lehenbauer
  - *Source*: [Silva & al., 2002]
- **Industrial sector**
  - *Definition*: Instead of speaking about *domain*, which is too general, we will speak about main industrial sectors related to safety-critical software. A LL is always linked with at least one industrial sector.
  - *Domain of values*: Aerospace, biomedical, defense, nuclear, transport... (extensible domain)
  - *Multiplicity*: 1-N
  - *Default value*: No [3]
  - *Example*: Aerospace
  - *Source*: [Silva & al., 2002]

[2] Name is a secondary identifier but there is also a technical primary identifier. We do not indicate all technical attributes in this section because it is not very useful.

[3] It could be filled automatically with the industrial sector entered by the user in his profile. However, it causes problems if he defines more than one industrial sectors in his profile.

- **Life cycle stage**
  - *Definition*: Instead of speaking about *"business processes"* and *"organization role"*, it is more convenient in software engineering to speak about main life cycle stages. It describes the phase(s) that was not performed properly and finally led to the reported problem. However, we decided that the multiplicity of this attribute is 0-N because it is not always easy to determine to which life cycle stage a LL is linked.
  - *Domain of values*: System requirements definition, system design, software requirements definition, software design, human-interface design, implementation, testing, deployment, usage, decommission, maintenance... (extensible domain)
  - *Multiplicity*: 0-N
  - *Default value*: No
  - *Example*: Software design, implementation
  - *Source*: [Silva & al., 2002]
- **Consequence**
  - *Definition*: Kind of damage provoked. It is important to add this attribute.
  - *Domain of values*: Deaths, resource losses, environmental damage, risks to lives... (extensible domain)
  - *Multiplicity*: 1-N
  - *Default value*: No
  - *Example*: Resource losses, risks to life
  - *Source*: [Silva & al., 2002]
- **Relevant sources**
  - *Definition*: Sources possessing some knowledge related to the LL.
  - *Domain of values*: Persons, books, article...
  - *Multiplicity*: 0-N
  - *Default value*: No
  - *Example*: The Risk Digest, Volume 9, Issue 88, 2 May 1990
  - *Source*: [Silva & al., 2002]
- **Proficiency level**
  - *Definition*: Level of proficiency at which the LL is available to the users.

- *Domain of values*: High, medium, low
- *Multiplicity*: 0-1
- *Default value*: No
- *Example*: High
- *Source*: [Silva & al., 2002]

- **Stability**
  - *Definition*: Frequency measure of the change of the LL content. This attributes is *"dead"* in our database because LL versioning is not currently performed in our prototype. It represents the number of times a LL has been changed, once it has been accepted by experts.
  - *Domain of values*: Integer
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: 3
  - *Source*: [Silva & al., 2002]

- **Creation date**
  - *Definition*: The original *"time"* attribute has no reason to exist in a LL repository. Indeed, a LL is directly available once it is posted and validated in the system. However, another time attribute is needed, which is the date of LL introduction in the system.
  - *Domain of values*: A valid date
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: 09/05/1998

- **Attachment**
  - *Definition*: Downloadable attachment, so it can be directly viewed by a user who consults a lesson. An attachment is characterized by a physical form that takes his values in the domain below.
  - *Domain of values*: Video, report, manual, chart... (extensible domain)
  - *Multiplicity*: 0-N
  - *Default value*: No
  - *Example*: Shuttle_video.avi

- **Related LL**
  - *Definition*: Cross-references to related LL stored in the repository. It could be interesting to add semantics to these references, i.e. different levels of similarity between LL could be established (resemblance, consolidation, contradiction with another LL).
  - *Domain of values*: Existing LL
  - *Multiplicity*: 0-N
  - *Default value*: No
  - *Example*: Shuttle crash at T+12 seconds after launching (Resemblance and consolidation).
  - *Source*: [Silva & al., 2002]
- **Related products and services**
  - *Definition*: References to other products and services that bear any relation to the LL. This attribute suggests that the submitted lesson can have impact on these products and services.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 0-N
  - *Default value*: No
  - *Example*: Hubble Space Telescope Website
  - *Source*: [Silva & al., 2002]
- **History**
  - *Definition*: Change history, management of the different versions of the LL, from its origin to the current version. Each time a user refines or modifies a LL, a new version of the LL is created. For example, the validation of a LL is generally performed in several steps, because there are interactions between the expert and the submitter of the LL. During this stage of validation, each time the LL is updated, a new version is created.
  - *Domain of values*: LL
  - *Multiplicity*: 1-N
  - *Default value*: Yes
  - *Source*: [Silva & al., 2002]
- **Statistics**
  - *Definition*: Statistics about the lessons, such as the reuse rate (number of feedbacks) of a lesson, the number of times it has been consulted or searched, can be helpful to users when they consult a specific lesson in the system.

- *Domain of values*: Integer
- *Multiplicity*: 1-1
- *Default value*: Yes
- *Example*: Search number: 1987, Consultation number: 156, Reuse rate: 23
- *Source*: [GAO, 2002]

5.2.2 Accident structure and accident event sequence

Each LL is existing because an accident happened that led to the creation of the lesson. There are several different accident models in risks analysis, project management and software engineering. Some are flexible and adaptable in all organizations, and others are used only in specific organizations. As said in [Silva & al., 2002], *"Any organization could structure its LL repository according to any other accident model that it considers to better suit its purpose. For example, there are accident models designed for the transport industry that can perform better than our proposal, at least for this industrial sector. As software is used in many industrial sectors, the analysis of software-related mishaps in one industry (e.g., transport) can suggest improvements for other industries (e.g., nuclear). Taking this into account, our model will take a more general approach, based on the well-known concepts of* ***hazard*** *and on the* ***events*** *that lead to an accident, which are not particular to any industrial sector".*

The accident describes the structure of the reported problem and contains the following attributes:

- **Accident name**
  - *Definition*: Label that summarizes the accident. It is important to warn the submitter of a LL that accident name should be different from LL name.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: Shuttle countdown hold.
  - *Source*: [Silva & al., 2002]
- **Accident date**
  - *Definition*: Date when the accident happened.
  - *Domain of values*: Valid date

- *Multiplicity*: 1-1
- *Default value*: No
- *Example*: 24/04/1998

- **Accident description**
  - *Definition*: Description of the accident.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: A software problem caused three minutes hold at T-31 during the launch countdown of the shuttle mission that orbited the Hubble Space Telescope on April 24th.
  - *Source*: [Silva & al., 2002]

- **Accident hazard**
  - *Definition*: Precursor of the accident. *"A hazard is a state that, given certain conditions, will inevitably lead to an accident. For example, if the distance between two planes is less than a required minimum, we have a hazard but not an accident (yet)"* [Silva & al., 2002].
  - *Domain of values*: Undefined string
  - *Multiplicity*: 0-1
  - *Default value*: No
  - *Example*: System reconfiguration was not enabled.
  - *Source*: [Silva & al., 2002]

- **Accident triggering conditions**
  - *Definition*: By definition, a hazard alone does not cause an accident. Other conditions are needed to lead to the accident. This attribute captures these conditions. In summary, an accident happens because there is a hazard and some triggering conditions occur.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 0-1 [4]
  - *Default value*: No
  - *Example*: Wrong code modification and the simulator could not find error.

[4]Multiplicity is 0-1 because it is a free-text field, so it could contain more than one triggering condition.

- *Source*: [Silva & al., 2002]

- **Accident risk [1-1]**: Risk is the product of the possibility of the hazard occurring and the magnitude of the worst consequence. A classification of risks is needed (in a range of tolerance) to decide which kind of measures should be taken. Accident risk encapsulates:

  - **Risk consequence**
    - *Definition*: Kind of consequence and damage done by the occurrence of the accident.
    - *Domain of values*: Catastrophic, marginal, perceptible, critical... (extensible domain)
    - *Multiplicity*: 1-1
    - *Default value*: No
    - *Example*: Critical
    - *Source*: [Silva & al., 2002], [Silva, 2003]
  - **Risk possibility**
    - *Definition*: Possibility/probability that the accident occurs.
    - *Domain of values*: High, low, medium
    - *Multiplicity*: 1-1
    - *Default value*: No
    - *Example*: Medium
    - *Source*: [Silva & al., 2002], [Silva, 2003]
  - **Risk justification**
    - *Definition*: Justification, explanation about the selected risk consequence and possibility.
    - *Domain of values*: Undefined string
    - *Multiplicity*: 1-1
    - *Default value*: No
    - *Example*: This explosion could have caused death of people and if new procedures are not respected, other problems with shuttle launching can happen again.
    - *Source*: [Silva & al., 2002]

- **Accident event sequences [0-N]**: An accident can be explained by one or more sequences of events that encapsulate all the chronological steps leading to the accident. An accident is described by [0-N] event sequences because it might be unknown. Each event sequence contains the following attributes:
  - **Accident event sequence type**
    - *Definition*: Type/form of the representation used to describe the event sequence. If the same sequence is described with several event sequence types, it must be the same source that relates it (see below).
    - *Domain of values*: Natural language narrative, cause consequence diagram, temporal logics... (extensible domain)
    - *Multiplicity*: 1-1
    - *Default value*: No
    - *Example*: Natural language narrative
    - *Source*: [Silva & al., 2002]
  - **Accident event sequence description**
    - *Definition*: Description step by step of the events that precede the accident.
    - *Domain of values*: Undefined string
    - *Multiplicity*: 1-1
    - *Default value*: No
    - *Example*: At T-48 seconds, newly written software detected that the outboard external tank liquid oxygen fill and drain valve was open when it should have been closed. The ground launch sequencer (GLS) stopped the countdown clock at T-31 seconds.
    - *Source*: [Silva & al., 2002]
  - **Accident event sequence source**
    - *Definition*: Description of the source (name and role) that relates the accident. It can be a witness, a passenger, a pilot, an injured person...
    - *Domain of values*: Undefined string
    - *Multiplicity*: 1-1 [5]
    - *Default value*: No
    - *Example*: Jean Aimar, supervisor of shuttle software systems.

[5] Multiplicity is 1-1 because it is a free-text field, so a same accident can be described by more than one source.

- *Source*: [Silva & al., 2002]

* **Accident event sequence discrepancies**
  - *Definition*: Disagreements and contradictions between the different sources.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 0-1
  - *Default value*: No
  - *Example*: Jean Aimar said that the shuttle countdown stopped at T-31 seconds, while an astronaut said that the instrument panel of the shuttle indicated T-29 seconds when the countdown stopped.
  - *Source*: [Silva & al., 2002]
* **Accident event sequence accuracy**
  - *Definition*: Level of certainty that users can have in the description of the accident event sequence. This level is important because it is sometimes very difficult to relate with accuracy an event sequence.
  - *Domain of values*: High, medium, low
  - *Multiplicity*: 0-1
  - *Default value*: No
  - *Example*: Low
  - *Source*: [Silva & al., 2002]

5.2.3 Solution

A LL must at least contain a solution or recommendation [1-N]. It indicates what should be done when some conditions happen. A solution is the means that allows to avoid the repetition of similar accidents in the future. This is essential because several systems, as the comp.risks forum, report lessons but do not give any solution to the reported problems. This implies that a LL system is complementary with comp.risks but does not completely replace it. Indeed, comp.risks contains also best practices, accident reports or alerts. Therefore, it could be interesting to discuss a problem in comp.risks before to submit it in the LL system. It could allow to warn interested people earlier, while information is not yet complete and structured.

A LL can of course be solved by many solutions. This is why users are allowed to post new solutions on a lesson submitted before. A solution contains the following attributes:

- **Author's identity**
  - *Definition*: User who posts the solution is not necessarily the user who posts the lesson, because a lesson can have many solutions. Of course, the user who posts the first found solution is the user who posts the lesson.
  - *Domain of values*: Undefined string (existing user)
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: Karl Lehenbauer
  - *Source*: [Silva & al., 2002]
- **Expert's identity**
  - *Definition*: Expert who checks the solution. More details about experts and validation are given in subsection 5.2.6. If it is the first solution of the lesson, the expert also checks the lesson itself. If it is an additional solution, expert only checks the solution because the lesson to which it is attached has been already validated.
  - *Domain of values*: Undefined string (existing expert)
  - *Multiplicity*: 0-1 (0 if the LL is submitted but not yet checked)
  - *Default value*: Yes
  - *Example*: Simon Defat, expert in shuttle systems.
  - *Source*: [Silva & al., 2002]
- **Hazard dealing type**
  - *Definition*: According to [Leveson, 1995], there is a hierarchy for dealing with a hazard in safety engineering. The purpose of hazard analysis is (1) to identify the hazards (i.e. the unsafe states) of the system under consideration, (2) evaluate the risks of the hazards and (3) identify measures that can be taken to eliminate or control the hazard, or to reduce the risk. In addition to reducing the hazards posed by a system, a secondary benefit of hazard analysis is that tradeoffs involving safety are made explicit and traceable. Concerning hazards related to accidents contained in the LL repository, potential solutions should be classified according to the hierarchy below. It is assumed that, generally, hazard elimination is the most difficult option, and damage reduction is the easiest one.

- *Domain of values*:
  - * *Hazard elimination*: Taking measures to make impossible the happening of the hazard. If successful with respect to all hazards, it means for the system to be intrinsically safe, i.e. that the system is not capable of doing any significant damage even in the case of the worst conceivable failure. If a system can be designed in this way (which is of course not possible for all systems) this is the safest option, and such a system is not safety-critical.
  - * *Hazard reduction*: Taking measures to reduce the **frequency** with which the hazard is expected to occur. Hazard reduction is similar to fault tolerance techniques, where local failures are contained without leading to system failures.
  - * *Hazard control*: Taking measures, if the hazard do occur, to reduce the **likelihood** of the hazard leading to an accident, i.e. reduce the likelihood of triggering conditions. *Fail-safe designs* are examples of such measures. It means that, in the event of a certain class of faults, the system will automatically go into a safe state. The emphasis is on safety and damage limitation. Continue functionality is not a priority. Hazard control reduces the severity of failures, by weakening the link between failures and accidents.
  - * *Damage reduction*: Taking measures to reduce the severity of the accident, i.e. the damage and losses caused by this accident.
- *Multiplicity*: 0-1
- *Default value*: No
- *Example*: Hazard control
- *Source*: [Silva & al., 2002], [Leveson, 1995]

- **Description**
  - *Definition*: Complete and precise description of the solution.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: A train vacuum brake is mentioned as an example of hazard control: if the pipe fails then the loss of the vacuum applies the brakes. Railway signaling systems are designed so that, in the event of failure, all trains should stop. This example illustrates that fail-safe mechanisms increases safety but not reliability.

- *Source*: [Silva & al., 2002], [Terry, 1991]

- **Creation date**
  - *Definition*: When the solution has been submitted in the system.
  - *Domain of values*: Valid date
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: 09/05/1998

- **Validation status**
  - *Definition*: The attribute *"nature"* of a general LL system is very vague. It is clearer to speak about *"validation status"*, which characterizes the LL solution in terms of quality and accuracy.
  - *Domain of values*: Heuristic, formal, complete, under development...
    (extensible domain)
  - *Multiplicity*: 0-1
  - *Default value*: No
  - *Example*: Heuristic
  - *Source*: [Silva & al., 2002]

- **Safety degree**
  - *Definition*: Degree of safety of the solution. Solutions to a problem are not unique, and different solutions provide different degrees of safety.
  - *Domain of values*: High, medium, low
  - *Multiplicity*: 0-1
  - *Default value*: No
  - *Example*: High
  - *Source*: [Silva & al., 2002]

- **Priority descriptor**
  - *Definition*: Priority level at which the solution should be applied and reused. Denotation of the risk, immediacy, and urgency of the solution content.
  - *Domain of values*: High, medium, low
  - *Multiplicity*: 0-1
  - *Default value*: No

- *Example*: Low
- *Source*: [van Heijst & al., 1997], [GAO, 2002]

- **Justification**
  - *Definition*: Method used to check the solution.
  - *Domain of values*: Proof, empirical, witness confirmation... (extensible domain)
  - *Multiplicity*: 0-1
  - *Default value*: No
  - *Example*: Empirical
  - *Source*: [Silva & al., 2002]

- **Uncertainty**
  - *Definition*: Rate of non confidence in the justification above.
  - *Domain of values*: High, medium, low
  - *Multiplicity*: 0-1
  - *Default value*: No
  - *Example*: High
  - *Source*: [Silva & al., 2002]

- **Test**
  - *Definition*: Attribute indicating whether the solution has been already tested in the past and whether it is currently used in similar situations.
  - *Domain of values*: Yes, no
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: Yes
  - *Source*: [Silva & al., 2002]

5.2.4 Usage feedback and evaluation

Users should have the possibility to submit feedbacks [0-N] and evaluations [0-N] on each solution of a LL. Both feedbacks and evaluations are written only by users and are not validated by experts.

A **usage feedback** is a comment about a solution's reuse. It is applied to a solution by potential users, with the aim of encouraging reuse or not. The feedback can be positive and/or negative. Feedbacks are very helpful

for users who wish to reuse a solution. A feedback is more or less like a message submitted in a forum, where subscribers can reply and exchange their ideas. It contains the following attributes:

- **Author's identity**
  - *Definition*: User who posts the feedback.
  - *Domain of values*: Undefined string (existing user)
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: Marc Hungoal
  - *Source*: [Silva & al., 2002]
- **Subject**
  - *Definition*: General label that summarizes the feedback, as a subject in a forum.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: Solution reused successfully on *"FirstFootOnMars"* project
  - *Source*: [Silva & al., 2002]
- **Date**
  - *Definition*: When the feedback has been posted in the system.
  - *Domain of values*: Valid date
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: 09/09/1999
- **Place**
  - *Definition*: Site where the feedback was applied. It describes for example the project in which the solution was reused.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: NASA, California - Project concerning the launch of the first shuttle in the direction of Mars.
  - *Source*: [Silva & al., 2002]

- **Application time**
  - *Definition*: Moment when the feedback was applied.
  - *Domain of values*: Valid date
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: 01/09/1999
  - *Source*: [Silva & al., 2002]
- **Description**
  - *Definition*: How the feedback was applied, in which circumstances.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: In the *"FirstFootOnMars"* project, solution was applied as described in the solution. There is nothing more to say: all is perfect!
  - *Source*: [Silva & al., 2002]

An **evaluation** is just a comment about a proposed solution, feedback or other evaluation. It is completely different from a feedback because it does not tell anything on the solution's reuse. The user will pay less attention to an evaluation because it might just be an advice not sustained by any evidence. An evaluation is like a message submitted in a forum. It contains the following attributes:

- **Author's identity**
  - *Definition*: User who posts the evaluation.
  - *Domain of values*: Undefined string (existing user)
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: Pascal Lendrier
  - *Source*: [Silva & al., 2002]
- **Commented object**
  - *Definition*: Object to which the evaluation is linked.
  - *Domain of values*: Solution, feedback, evaluation
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: Evaluation (about the feedback above).
  - *Source*: [Silva & al., 2002]

- **Subject**
  - *Definition*: General label that summarizes the evaluation, as a subject in a forum.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: Asking for more information.
  - *Source*: [Silva & al., 2002]
- **Date**
  - *Definition*: When the evaluation has been posted in the system.
  - *Domain of values*: Valid date
  - *Multiplicity*: 1-1
  - *Default value*: Yes
  - *Example*: 29/09/1999
  - *Source*: [Silva & al., 2002]
- **Description**
  - *Definition*: Point of view of the submitter about a specific solution, feedback or evaluation.
  - *Domain of values*: Undefined string
  - *Multiplicity*: 1-1
  - *Default value*: No
  - *Example*: The description of your usage feedback is not very precise. Could you give more information about the *FirstFootOnMars* project?
  - *Source*: [Silva & al., 2002]

5.2.5 User profile

Throughout this document, we clearly mentioned that it is necessary and useful to use a user profile associated with each user as an aid to match the information contained in the repository with the potentially interested individuals. In order to receive automatically attractive lessons and to be identified, users have to fill a precise and complete profile, in which they define their areas of interest.

Active dissemination requires of course user profiles and is performed by matching common attributes of the user profiles with attributes of the LL

in the repository. Consequently, the list of these common attributes should be firstly established to index the LL. They can be the main attributes used in a manual and personal search (passive dissemination) in the system. A profile could be constructed according to a set of attributes, as industrial sectors, life cycle stages or consequences... Consequently, in order to define profiles, some attributes will have a higher priority to index the LL repository. These ones are directly related to the jobs of the workers. However, not all attributes are relevant to be included in a profile. *"The reason for adding these attributes to user profiles is that they can be used to classify both the lessons and the potential users, as each LL user has some responsibility in developing software in a particular* ***industrial sector*** *(industry), is focused on a particular stage of the* ***life cycle****, and is interested in the possible* ***consequences****. Any attempt to facilitate user-profiling should, at least, take into account these attributes. However, this list of attributes is not necessarily complete or adapted for every situation"* [Silva & al., 2002].

The suitable attributes of a user profile are enumerated below:

- **Personal and general information**: Personal information includes the identity of the user [1-1], his login/password [1-1], his country [1-1], his email [1-1], his job [0-1], the company or organization where he works [0-1], his contact numbers [0-1], his choice to reveal or not his identity [1-1]. Personal information allows to identify the user in the system and to learn a little more about his skills.

- **Profile information**: If the user chooses to receive lessons by active dissemination, he has to fill profile information. It concerns the main common attributes of the profiles and the lessons, e.g. industrial sectors, life cycle stages, consequences, interested products and services, proficiency level and types of attachment. At least one of these attributes must be selected by the user if he wants to receive lessons by active dissemination. In the other case, they are all optional.

These profiles also help to know what the knowledge and skills of the users are. Users should obviously have the possibility to **update** their profiles, so they can manage the amount and type of information they receive.

5.2.6 Expert

Experts are playing the most important role in the system. They accomplish the verification step of the LL process, which consists in evaluating and validating all lessons and solutions submitted in the repository for relevance, correctness, non-redundancy and consistency. Experts are needed to perform this task and their roles go beyond those of the moderators in forums like comp.risks. They have to take decisions about **accepting**, **modifying** or **rejecting** all new LL and new solutions on existing LL.

Once they are stored in the repository, lessons and solutions are reviewed and checked by technical experts before being disseminated and made accessible to users. When the user submits a LL, most attributes are optional because he could not understand their meanings. This implies that experts have to check and determine most of these optional attributes in order to refine the LL as most as possible. Experts can also update and create **links between LL**. Before validating a lesson, a **dialog** between them and the submitter of the LL is most of the time necessary. This dialog allows experts to ask more questions and more precise information to the users. Therefore, a **profile** will also be needed to identify experts and to define their areas of knowledge and specialization. For example, it is possible to imagine a classification of experts based on the industrial sectors (domains) in which they are specialists.

Chapter 6

Features of lessons learned systems

As explained in chapter 4, an iterative process is used. Each **iteration** is a new step towards the final system. Only the essential features of a LL system have been considered, as the important goal was to have a rapid overview of the implementation results. A **priority** has been assigned to the features, so that successive iterations deal with features of decreasing priority level.

This chapter is partially linked with a later part about future works and perspectives. Features presented in this chapter are those which were analyzed and/or implemented during the imparted time to build the prototype, i.e. four months. Other possible features are enumerated in part V about future works.

Features have been divided in six groups. The five first groups match with the five tasks of the **LL process**: collection, verification, storage, dissemination, and reuse. The last group concerns the administration functions.

The list of the features below were firstly considered to design the prototype. Each feature has an alphanumeric identifier.

6.1 Collection

Only **passive collection** is used to gather lessons. Passive collection is easier to implement than active collection and is better suited to our purpose. In the last part of this work, possibilities and advantages of active collection, with the comp.risks forum, are presented. Two forms of passive collection are distinguished.

- **Passive collection of a new lesson (F1.1)**
  The user submits his own LL to the system through a web form.

- **Passive collection of a new solution on an existing lesson (F1.2)**
  If a user finds a better solution concerning a problem submitted earlier, he can post it to bring more possibilities about solving the encountered problem.

6.2 Verification

As seen before, a LL repository should be evaluated and validated by experts for relevance, correctness, non-redundancy and consistency.

- **Manual verification of non-redundancy (F2.1)**
  Each time a lesson is submitted, it is validated by an expert moderator for non-redundancy. This expert can check it by performing a search in the repository, to see if other similar LL exist in the system. If some lessons are very close, he should have the possibility to create references between them, as well as create a new lesson which mixed the similar lessons or remove one of them if a LL includes another LL.

- **Computer-aided verification of non-redundancy (F2.2)**
  After a submission, the system can automatically search for lessons that have direct relation with the current lesson. So, when experts check lessons for the first time, they can directly see probable and relevant related lessons.

- **Verification of consistency, correctness and relevance (F2.3)**
  The expert moderator checks the consistency, the correctness and the relevance of the new lesson or the new solution submitted on an accepted lesson. Afterwards, he accepts, modifies or rejects the new lesson/solution. During this time, he can also have an interactive dialog with the submitter, to ask more details and to have a complete exchange of information. If he decides to reject the lesson/solution, he should justify his refusal and the submitter must be notified. After posting and during the validation, the user can list all the lessons/solutions he has submitted and view the verification status of each of them. He can also answer to the comments of the expert moderator.

- **LL versions (F2.4)**
  Lessons can be continuously updated during and after the validation. An history of the different versions of the lessons would be helpful.

6.3 Storage

Storage concerns the physical representation, indexation and structure of the database. Therefore, it does not really contain user-oriented features.

6.4 Dissemination

Users can retrieve or receive lessons by two different ways, called passive and active dissemination.

- **Profile management (F4.1)**
  Users can create, modify or delete their profiles in which they define their areas of interest. Profiles are used in active dissemination and to identify users.

- **Passive dissemination**
  The user retrieves himself lessons on the system in one of the following manners:

  - **Hierarchical search (F4.2)**
    The user searches for lessons by browsing through a web form in which lessons are hierarchically classified by themes. This kind of search is not very useful in accidents related to critical software, mainly because it is very difficult to establish a classification following our attributes topology. It would also be redundant with the search by attributes and keywords.
  - **Search by attributes (F4.3)**
    The user searches for lessons by filling a web form that contains fixed attributes of the lessons.
  - **Search by keywords (F4.4)**
    The user searches for lessons by typing a list of keywords in a kind of web search engine, such as google. He can define the scope of his search.
  - **Integration with external systems (F4.5)**
    It could be useful to see if it is possible to integrate the LL system with current applications, as Microsoft Word and Microsoft Excel. For example, it could be done by using macros. The user selects a text and then clicks on a macro button. The macro then links the selected text with the search by keywords of the system. The browser of the user starts with a screen showing the search results and he can select lessons corresponding to his search. An API could also be available if someone wants to write a macro for another application.

- **Active dissemination with user profiles (F4.6)**
  Active dissemination allows users to be reminded about lessons without any solicitation. Users are notified by email of lessons they are interested in. It requires user profiles, defining their areas of interests.

6.5 Reuse

Evaluation or feedback about a LL helps future users in a better use or understanding of a solution.

- **Evaluation about reported solutions, feedbacks or evaluations (F5.1)**
  Each solution can be commented by users. The user can also comment feedbacks and existing evaluations.

- **Usage feedback about applied solutions (F5.2)**
  After a solution has been applied in practice by a user, the system allows the user to record a feedback on the outcome. The feedback includes also the place, the time and the description of the reuse.

6.6 Administration

- **Fixed attribute value management (F6.1)**
  The administrator can add or remove new values of fixed attributes in the database, as industrial sector or life cycle stage values for example.

- **Expert management (F6.2)**
  The administrator can give/remove rights to a normal user/expert in order to become an expert/normal user.

6.7 Summary of the features

The table below shows, for each feature, its priority, whether it was analyzed and implemented, and in which iteration (version), if any. There were **four iterations**.

Id.	Feature	Priority	Implemented	Version
	Collection			
F1.1	Passive collection about a new lesson	High	Yes	1
F1.2	Passive collection about a new solution	Medium	Yes	2
	Verification			
F2.1	Non-redundancy (manual)	Medium	Yes	2
F2.2	Non-redundancy (computer-aided)	Low	No	/
F2.3	Consistency, correctness and relevance	Medium	Yes	2
F2.4	LL versions	Low	No	/
	Storage			
	Dissemination			
F4.1	Profile management	High	Yes	1
F4.2	Hierarchical search	Low	No	/
F4.3	Search by attributes	High	Yes	2
F4.4	Search by keywords	High	Yes	2
F4.5	Integration with external systems	Low	Yes	4
F4.6	Active dissemination with profiles	Medium	Yes	3
	Reuse			
F5.1	Evaluation	Medium	Yes	3
F5.2	Usage feedback about applied solutions	Medium	Yes	3
	Administration			
F6.1	Fixed attribute value management	Low	Yes	4
F6.2	Expert management	Low	Yes	4

Table 6.1: Summary of the features.

Chapter 7

Class diagram of the lessons learned system

According to the list of attributes defined in section 5.2, we drafted the following class diagram.

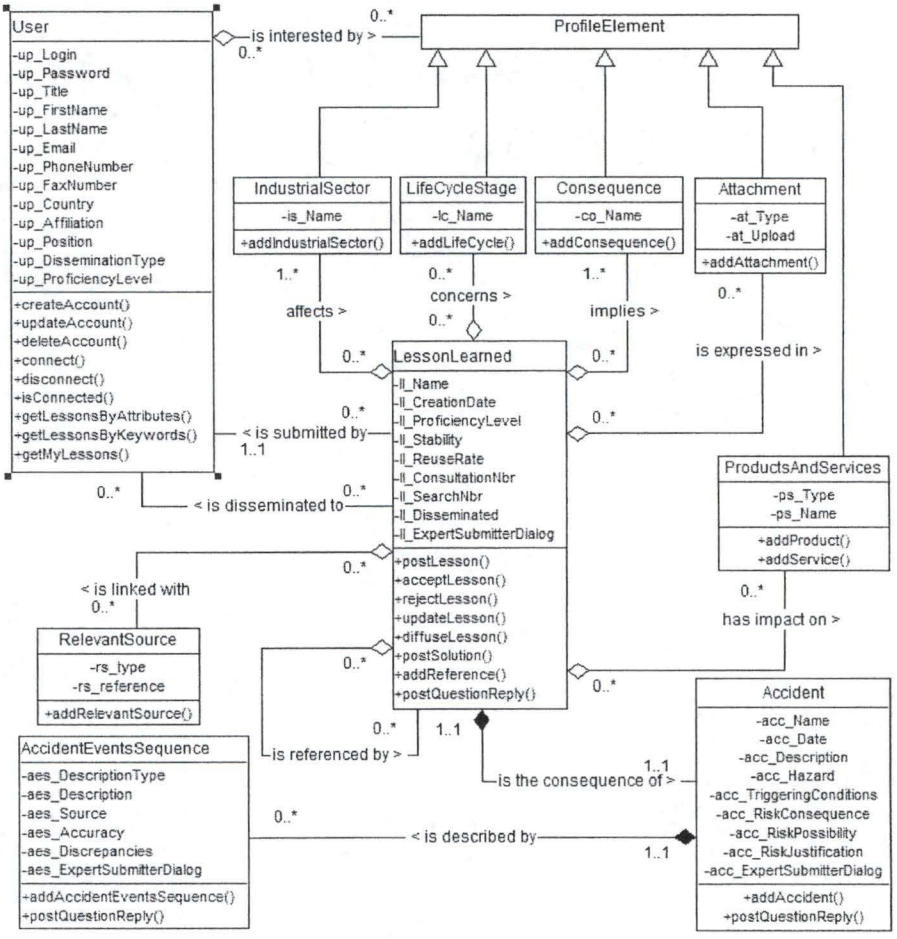

Figure 7.1: Class diagram - Part 1.

In figure 7.2, there is an *"exactly-1"* exclusion constraint between the class *Evaluation* and the three roles *"is related to"* played by this class.

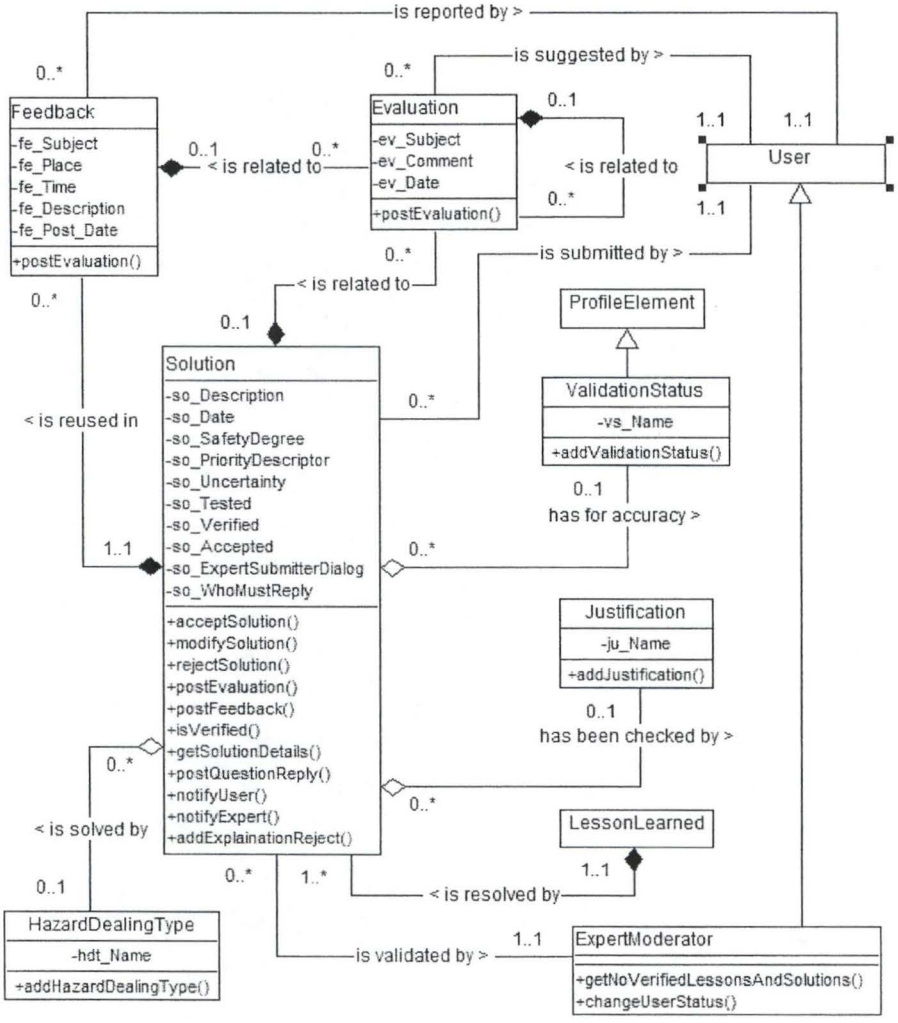

Figure 7.2: Class diagram - Part 2.

Chapter 8

Use cases of the lessons learned system

8.1 Use cases schema and description

A description of the main Use Cases (UC) is given in this chapter. We split the UC in two categories, according to [Cockburn, 2000], depending on the level of their goal:

- **User goal level** is *"the goal the primary actor has in trying to get work or the one the user has in using the system"*.
- **Subfunction level** are *"goals required to carry out user goals"*.

In the following description of UC, we give a summarized description of the subfunction level goals. Only UC that have a user goal level are described in detail. Several UC are represented in more than one schema, but they are defined only once.

8.1.1 Use cases related to profile management

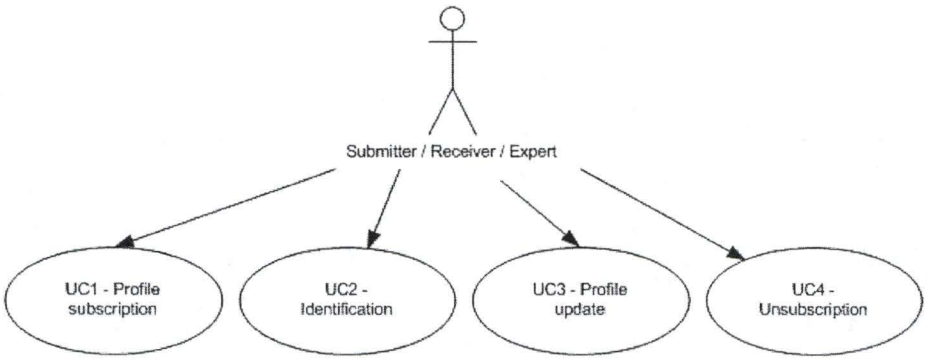

Figure 8.1: UC related to profile management.

Profile subscription (UC1)

Description	The user records his profile to have authorized access in the system. Profile is used to identify the user and it allows him to have access to active dissemination. The user enters general information about himself (login, password, first name, last name, email...) and information about lessons in which he is interested (industrial sectors, life cycle stages, consequences, products and services, type of attachment, validation status, proficiency level).
Level	Subfunction level.

Identification (UC2)

Description	The user enters his login/password to be connected to the LL system.
Level	Subfunction level.

Profile update (UC3)

Description	The user updates his profile information given in UC1.
Level	Subfunction level.

Unsubscription (UC4)

Description	It concerns unsubscription related to active dissemination. The user asks not to receive automatically any lessons in the future.
Level	Subfunction level.

8.1.2 Use cases related to collection and reuse

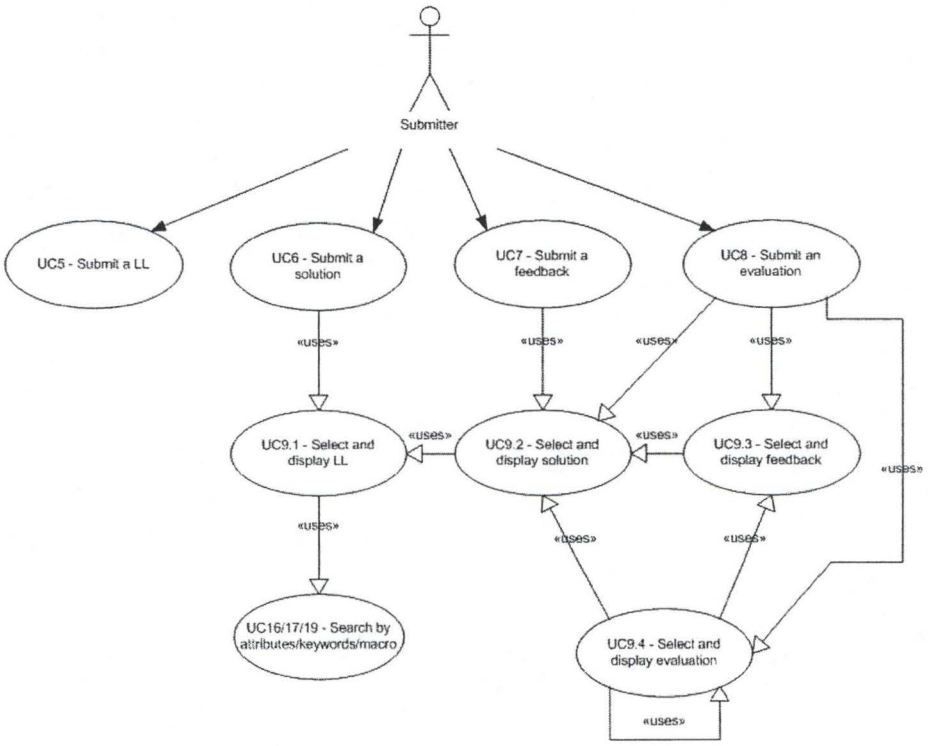

Figure 8.2: UC related to collection and reuse.

Submit a LL (UC5)

Description	The user submits a new LL. He provides the causes of the encountered problem, and the recommended solution he used.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2).
Trigger	The user clicks on the *"LL submission"* button.
Main success scenario	1. The user enters the LL name [1-1], industrial sector(s) [1-N], life cycle stage(s) [0-N], consequence(s) [1-N], relevant source(s) [0-N], product(s) and service(s) [0-N], proficiency level [0-1] and downloads attachment(s) [0-N]. 2. He enters information about the accident: name [1-1], date [1-1], description [1-1], hazard [0-1] and triggering conditions [0-1]. 3. He enters information about risk: consequence [1-1], possibility [1-1] and justification [1-1]. 4. He enters accident event sequence(s) [0-N]. 5. He enters the recommended solution: hazard dealing type [0-1], description [1-1], safety degree [0-1], priority descriptor [0-1], validation status [0-1], justification [0-1], uncertainty [0-1], tested solution [1-1]. 6. The system confirms that the LL and solution have been posted successfully. LL and solution are not accessible and disseminated to users because they have not been yet validated by experts.
Extensions	1a. While the user does not properly enter LL name, at least one industrial sector, and at least one consequence, the system asks it again. 2a. While the user does not properly enter accident name, date and description, the system asks it again. 3a. While the user does not properly enter risk consequence, possibility and justification, the system asks it again. 5a. While the user does not properly enter solution description and indicates if it was tested before, the system asks it again.
Success guarantee	The LL and solution are recorded in the system.

Submit a solution (UC6)

Description	The user submits a new recommended solution about a validated LL.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The LL on which the user wants to submit a new solution has been accepted by an expert (UC11). - The user has selected a LL (UC9.1) after having performed a search in the system (UC16 or UC17 or UC19).
Trigger	The user clicks on the *"Post solution"* button.
Main success scenario	1. The user enters the recommended solution: type of the solution [0-1], description [1-1], safety degree [0-1], priority descriptor [0-1], validation status [0-1], justification [0-1], uncertainty [0-1], if the solution was tested before [1-1]. 2. The system confirms that the solution is recorded. The new solution is not accessible and disseminated to users because it has not been yet validated by experts.
Extensions	1a. While the user does not properly enter the description and indicates if the solution was tested before, the system will ask it again.
Success guarantee	The solution is recorded in the system.

Submit a feedback (UC7)

Description	The user submits a feedback about a validated solution he reused.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The solution on which the user wants to submit a feedback and the related LL have been accepted by an expert (UC11 and/or UC12). - The user has selected a LL and one of its solution (UC9.1 and UC9.2) after having performed a search in the system (UC16 or UC17 or UC19).
Trigger	The user clicks on the *"Post feedback"* button.
Main success scenario	1. The user enters feedback information: subject [1-1], place [1-1], time [1-1] and description [1-1]. 2. The system confirms that the feedback is correctly recorded.
Extensions	1a. While the user does not properly enter the subject and the description, the system will ask it again.
Success guarantee	The feedback is recorded in the system.

Submit an evaluation (UC8)

Description	The user submits an evaluation about a validated solution, a feedback, or an evaluation.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The solution on which the user wants to submit an evaluation and the related LL have been accepted by an expert (UC11 and/or UC12). - The user has selected a LL and one of its solution (UC9.1 and UC9.2) after having performed a search in the system (UC16 or UC17 or UC19). - Once he has selected a solution, the user may have selected a feedback or an evaluation (UC9.3 and/or UC9.4), but it is not obligatory.
Trigger	The user clicks on the *"Post evaluation"* button.
Main success scenario	1. The user enters evaluation information: subject [1-1] and comment [1-1]. 2. The system confirms that the evaluation is correctly recorded.
Extensions	1a. While the user does not properly enter the subject and the comment, the system will ask it again.
Success guarantee	The evaluation is recorded in the system.

Select and display LL (UC9.1)

Description	The user selects a LL he wants to see.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The user has performed a search in the system (UC16 or UC17 or UC19) or has asked to see his submitted LL (UC15). If the user is an expert, he could also have asked for all non verified LL (UC13).
Trigger	The user clicks on the LL link he wants to display.
Main success scenario	1. The users select the LL. 2. The system displays the corresponding LL.
Success guarantee	The corresponding LL is displayed.

Select and display solution (UC9.2)

Description	The user selects a solution he wants to see.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The user has selected a LL (UC9.1). - The user has performed a search in the system (UC16 or UC17 or UC19) or has asked to see his submitted LL/solutions (UC15). If the user is an expert, he could also have asked for all non verified LL/solutions (UC13).
Trigger	The user clicks on the solution link he wants to display.
Main success scenario	1. The users select the solution. 2. The system displays the corresponding solution.
Success guarantee	The corresponding solution is displayed.

Select and display feedback (UC9.3)

Description	The user selects a feedback he wants to see.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The user has selected a LL (UC9.1) and a solution (UC9.2) that have been validated by an expert (UC11 and/or UC12). - The user has performed a search in the system (UC16 or UC17 or UC19) or has asked to see his submitted LL/solutions (UC15).
Trigger	The user clicks on the feedback link he wants to display.
Main success scenario	1. The users select the feedback. 2. The system displays the corresponding feedback.
Success guarantee	The corresponding feedback is displayed.

Select and display evaluation (UC9.4)

Description	The user selects an evaluation he wants to see.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The user has selected a LL (UC9.1) and a solution (UC9.2) that have been validated by an expert (UC11 and/or UC12). He could also have selected a feedback or another evaluation (UC9.3 and/or UC9.4). - The user has performed a search in the system (UC16 or UC17 or UC19) or has asked to see his submitted LL/solutions (UC15).
Trigger	The user clicks on the evaluation link he wants to display.
Main success scenario	1. The users select the evaluation. 2. The system displays the corresponding evaluation.
Success guarantee	The corresponding evaluation is displayed.

8.1.3 Use cases related to verification

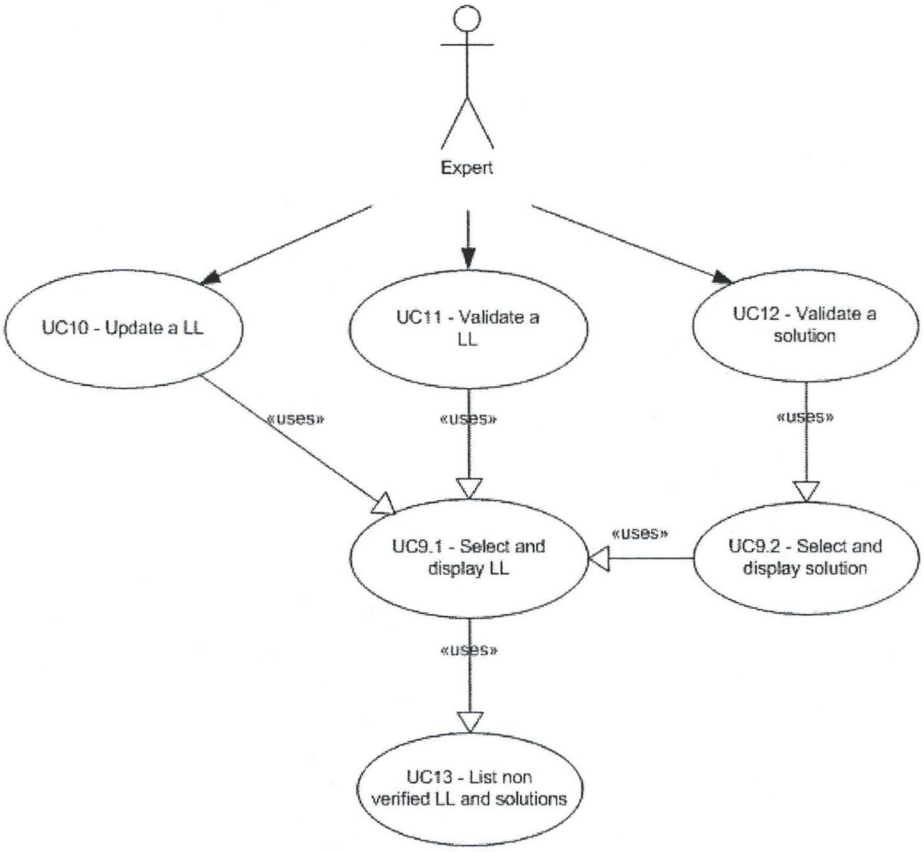

Figure 8.3: UC related to verification - Part 1.

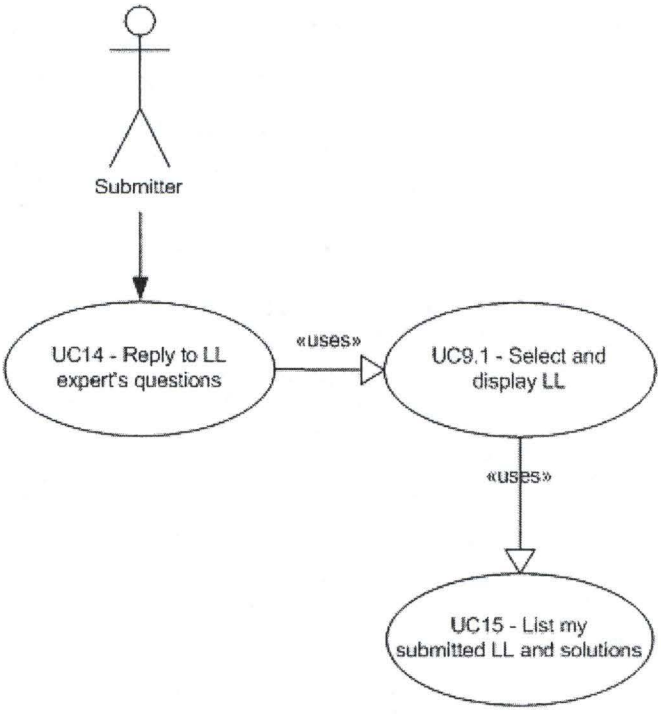

Figure 8.4: UC related to verification - Part 2.

Update a LL (UC10)

Description	The expert updates a LL that is not yet validated. He can ask questions and more information to the submitter of the lesson and solution, by recording personal comments for each part of the LL (LL general information, accident, event sequences and solution). Once the expert has asked questions and updated the LL, the LL submitter is notified by mail and he can see expert comments by consulting the LL and solutions he has submitted.
Level	User goal level.
Preconditions	- The expert is connected to the LL system (UC2). - The LL or a new solution related to the LL has been submitted (UC5 or UC6) and is not yet validated. - The expert has selected the LL (UC9.1) after having retrieved all the non verified LL and solutions (UC13).
Trigger	The expert clicks on the *"Record" (update)* or *"Notify"* button.
Main success scenario	1. The expert updates the fields of the LL and/or solution. 2. The expert enters and records comments (questions to the submitter) about LL general information, accident, event sequences and/or solution. 3. The expert notifies the user (an email is sent) that his LL/solution has been updated, i.e. that he wants more details about it.
Success guarantee	The LL and solution are updated.

Validate a LL (UC11)

Description	The expert accepts or rejects a submitted LL. The submitter of the lesson is notified.
Level	User goal level.
Preconditions	- The expert is connected to the LL system (UC2). - The LL has been submitted (UC5 or UC6) and is not yet validated. - The expert has selected the LL (UC9.1) after having retrieved all the non verified LL and solutions (UC13).
Trigger	The expert clicks on the *"LL accept"* or *"LL reject"* button.
Main success scenario	1. The expert accepts the LL. 2. The system notifies the user who posted the LL.
Extensions	1a1. The expert rejects the LL. 1a2. The expert enters the reasons why he is rejecting the LL. 1a3. The system notifies the user who posted the LL.
Success guarantee	The LL is accepted or rejected.

Validate a solution (UC12)

Description	The expert accepts or rejects a submitted solution on an existing lesson. The submitter of the solution is notified.
Level	User goal level.
Preconditions	- The expert is connected to the LL system (UC2). - The LL has been submitted (UC5) and accepted (UC11). - A new solution on the LL has been submitted (UC6) and is not yet accepted. - The expert has selected the LL (UC9.1) after having retrieved all the non verified LL and solutions (UC13).
Trigger	The expert clicks on the *"Accept solution"* or *"Reject solution"* button.
Main success scenario	1. The expert accepts the solution. 2. The system notifies the user who posted the solution.
Extensions	1a1. The expert rejects the solution. 1a2. The expert enters the reasons why he is rejecting the solution. 1a3. The system notifies the user who posted the solution.
Success guarantee	The solution is accepted or rejected.

List non verified LL and solutions (UC13)

Description	The expert asks for obtaining the non verified new LL and solutions related to his area of knowledge. He can see the state (if it needs modification by submitter or expert) of each lesson and solution, i.e. if submitters have replied to his questions concerning a specific lesson or solution.
Level	User goal level.
Preconditions	The expert is connected to the LL system (UC2).
Trigger	The expert clicks on the *"LL and solution validation"* button.
Main success scenario	1. List of non verified LL and solutions is displayed.
Extensions	1a. There is no new non verified LL and solutions, so the list is empty.
Success guarantee	The list is displayed.

Reply to LL expert's questions (UC14)

Description	In order to validate the LL or solution he submitted, the user replies to the questions asked by the expert.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2). - The user has been submitted a new LL or solution (UC5 or UC6) that is not yet accepted. - The expert has updated the LL and notified the user about his modifications and questions (UC10). - The user has selected a LL/solution (UC9.1 and/or UC9.2) after having asked the list of all LL and solutions he submitted (UC15).
Trigger	The user clicks one the *"Record" (update)* button.
Main success scenario	1. The user enters his reply to the questions of the expert. 2. The system notifies the expert.
Success guarantee	Answers to the questions are recorded and expert is warned.

List my submitted LL and solutions (UC15)

Description	The user asks to see all verified or non verified LL and solutions he posted. He can see the state (*"accepted"*, *"refused"*, *"needs modification by submitter"*, *"needs validation by expert"*) of each of them.
Level	User goal level.
Preconditions	- The user has recorded a profile (UC1). - The user is connected to the LL system (UC2).
Trigger	The user clicks one the *"My LL and solutions"* button.
Main success scenario	1. The list of his LL and solutions is displayed.
Extensions	1a. The list is empty, the user has never submitted any LL or solution.
Success guarantee	The list is displayed.

8.1.4 Use cases related to dissemination

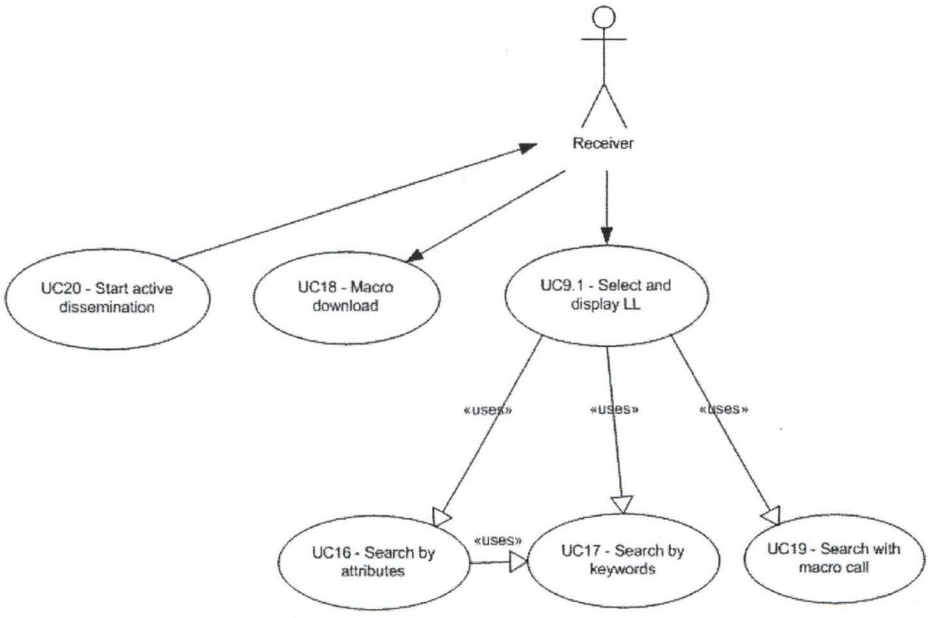

Figure 8.5: UC related to dissemination.

Search by attributes (UC16)

Description	The user searches for LL by giving values related to fixed attributes of the LL (for example, industrial sector, life cycle stage or consequence).
Level	Subfunction level.

Search by keywords (UC17)

Description	The user searches for LL by giving some keywords.
Level	Subfunction level.

Macro download (UC18)

Description	The user downloads and installs macros to integrate automatic call to the LL system within his current applications, as Microsoft Word and Microsoft Excel for example.
Level	Subfunction level.

Search with macro call (UC19)

Description	The user searches for LL, through his Microsoft Word or Excel applications.
Level	User goal level.
Preconditions	The user has downloaded a macro (UC18).
Trigger	The user clicks on the *"Search LL"* macro button.
Main success scenario	1. The user selects a text. 2. He clicks on the *"Search LL"* macro button. 3. The LL system connects the user and displays the corresponding LL.
Success guarantee	The corresponding LL are displayed.

Start active dissemination (UC20)

Description	Every time a LL is accepted by an expert, the lesson is automatically disseminated by mail to users whose profile matches with these lessons.
Level	User goal level.
Preconditions	The user has recorded a profile (UC1) in which he asks for active dissemination.
Trigger	An expert accepts a LL by pushing on the *"LL accept"* button (UC11).
Main success scenario	1. The system sends an email to users whose profile matches with the validated LL. 2. Users receive an email.
Success guarantee	The corresponding LL is sent.

8.1.5 Use cases related to administration

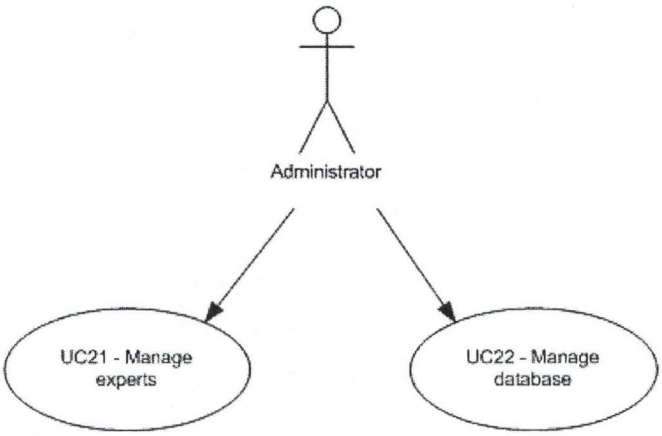

Figure 8.6: UC related to administration.

Manage experts (UC21)

Description	The administrator gives/removes rights to a normal user/expert to become an expert/normal user.
Level	User goal level.
Preconditions	The administrator is connected to the LL system (UC2).
Trigger	The administrator clicks on the *"Manage expert list"* button.
Main success scenario	1. The administrator enters the login of the user for whom he wants to change the rights. 2. The system displays the rights of this user. 3. The administrator changes his rights. 4. The system confirms the update.
Extensions	1a. The administrator enters a wrong login and the system asks it again.
Success guarantee	User rights are changed.

Manage database (UC22)

Description	The administrator adds or removes new values related to fixed attributes in the database, such as industrial sectors or life cycle stages values for example.
Level	User goal level.
Preconditions	The administrator is connected to the LL system (UC2).
Trigger	The administrator clicks on the *"Data manager"* button.
Main success scenario	1. The administrator enters new values or updates existing values related to fixed attributes in the database.
Success guarantee	Values related to fixed attributes in the database are updated.

8.2 Relation between use cases and features

	Features															
	1.1	1.2	2.1	2.2	2.3	2.4	4.1	4.2	4.3	4.4	4.5	4.6	5.1	5.2	6.1	6.2
UC1							X									
UC2	X	X	X		X		X		X	X			X	X	X	X
UC3							X									
UC4							X									
UC5	X															
UC6		X														
UC7														X		
UC8													X			
UC9									X	X	X					
UC10			X		X											
UC11			X		X											
UC12			X		X											
UC13			X		X											
UC14					X											
UC15					X											
UC16									X							
UC17										X						
UC18											X					
UC19											X					
UC20							X					X				
UC21																X
UC22															X	

Table 8.1: Relation between UC and features.

UC with a user goal level are represented in bold in the table 8.1.

Features 2.2, 2.4 and 4.2 have not been implemented. This should be done in a future iteration of the prototype. Consequently, it is logical they do not have any correspondence with UC.

Chapter 9

Database of the lessons learned system

It is impossible to represent on one page the normalized entity/relationship (E/R) diagram of the whole database. Hence, we first show a summarized global schema including only entities and relationships, without attributes. Secondly, this global schema is decomposed into four subschemas including attributes. Some entities are present in more than one subschema. The entities of the four subschemas are logically grouped: the first schema is centered around user profile, the second around LL, the third around solution / feedback / evaluation, and the last around interaction between user profile and the former three.

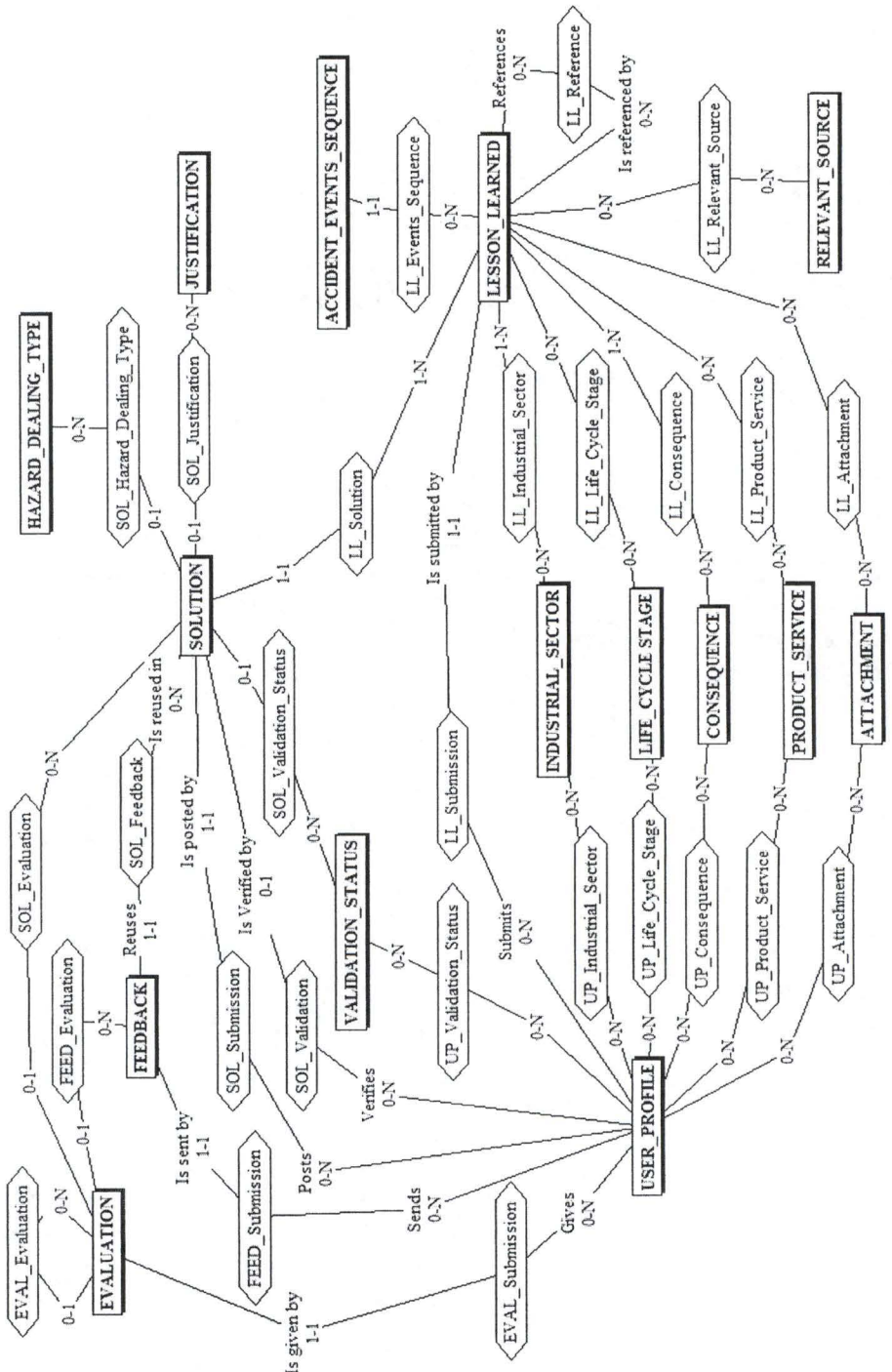

Figure 9.1: Summarized E/R diagram.

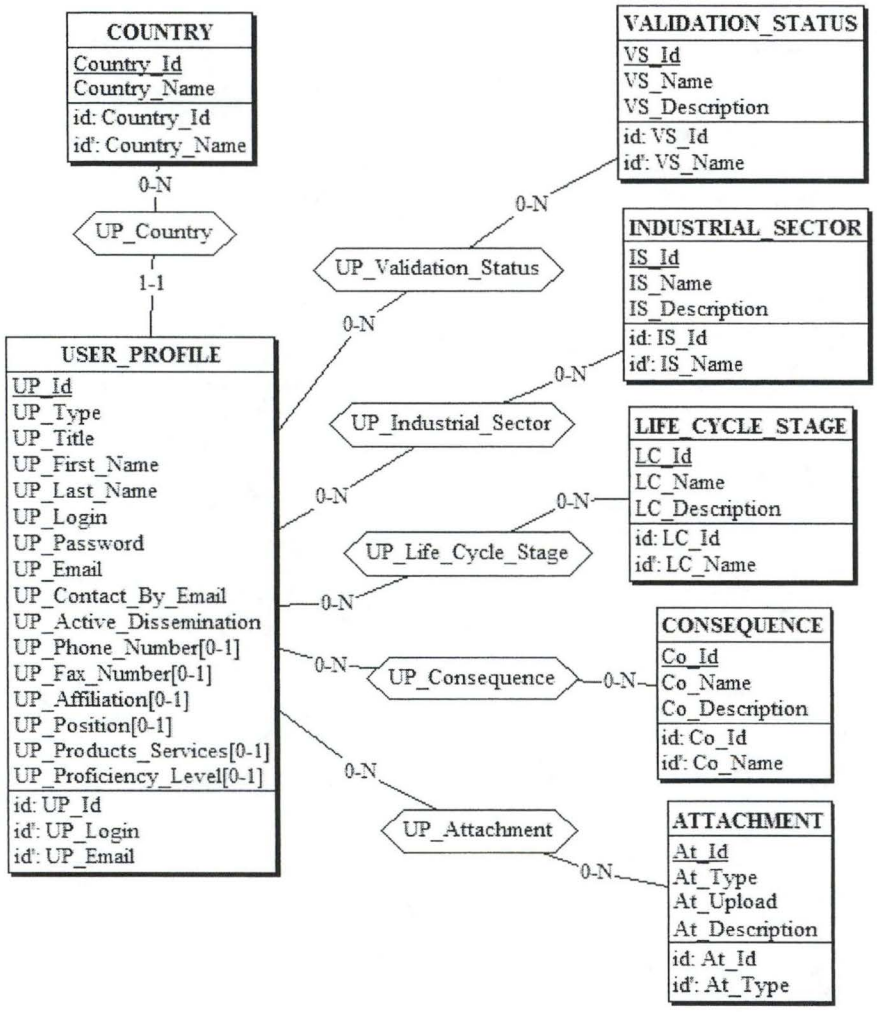

Figure 9.2: E/R subschema (1) - User profile.

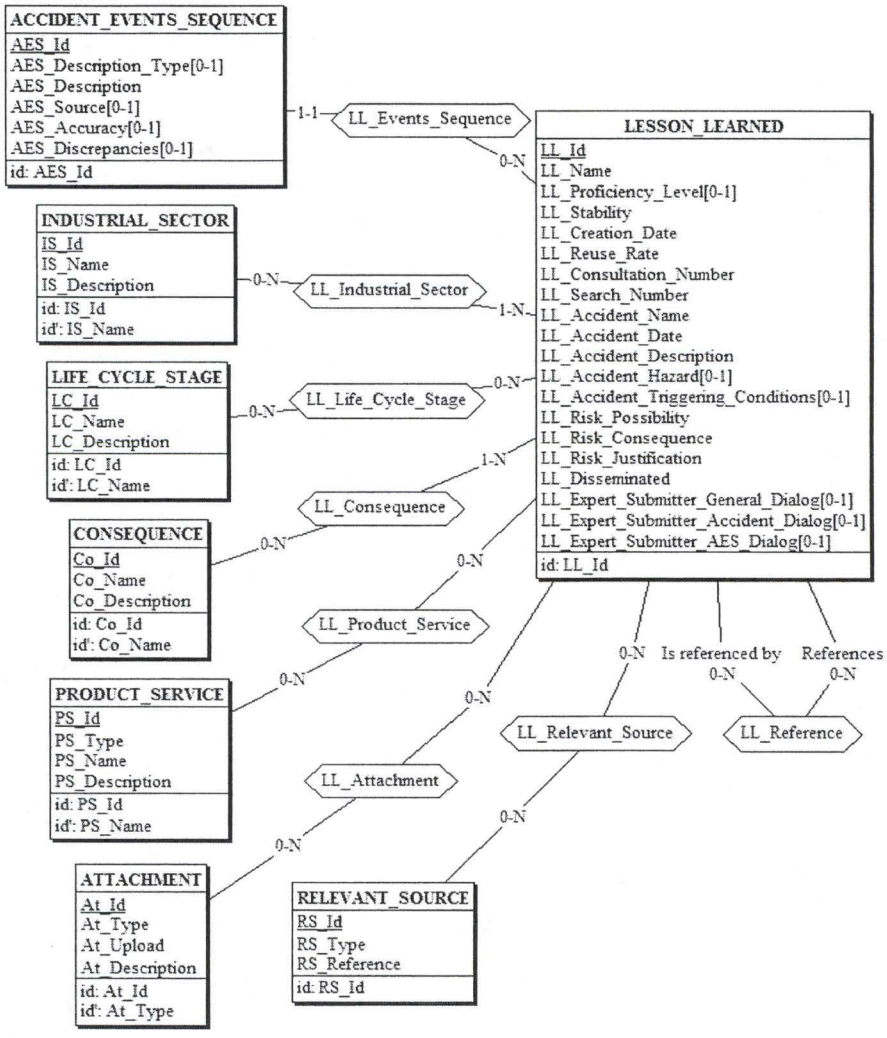

Figure 9.3: E/R subschema (2) - LL.

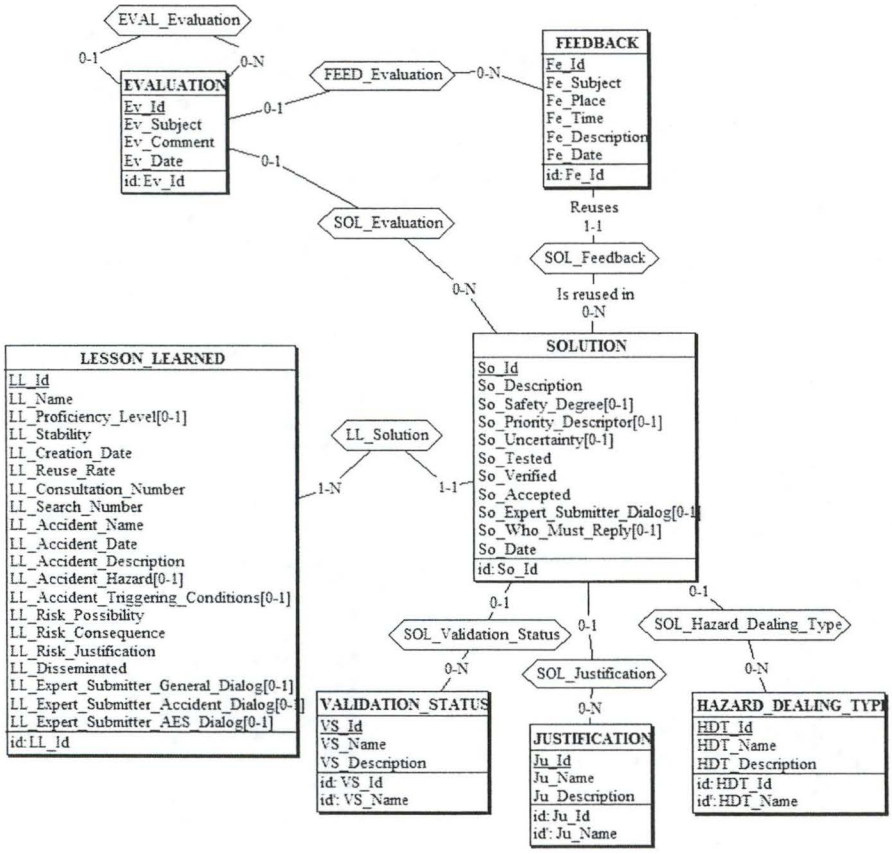

Figure 9.4: E/R subschema (3) - LL / solution / feedback / evaluation.

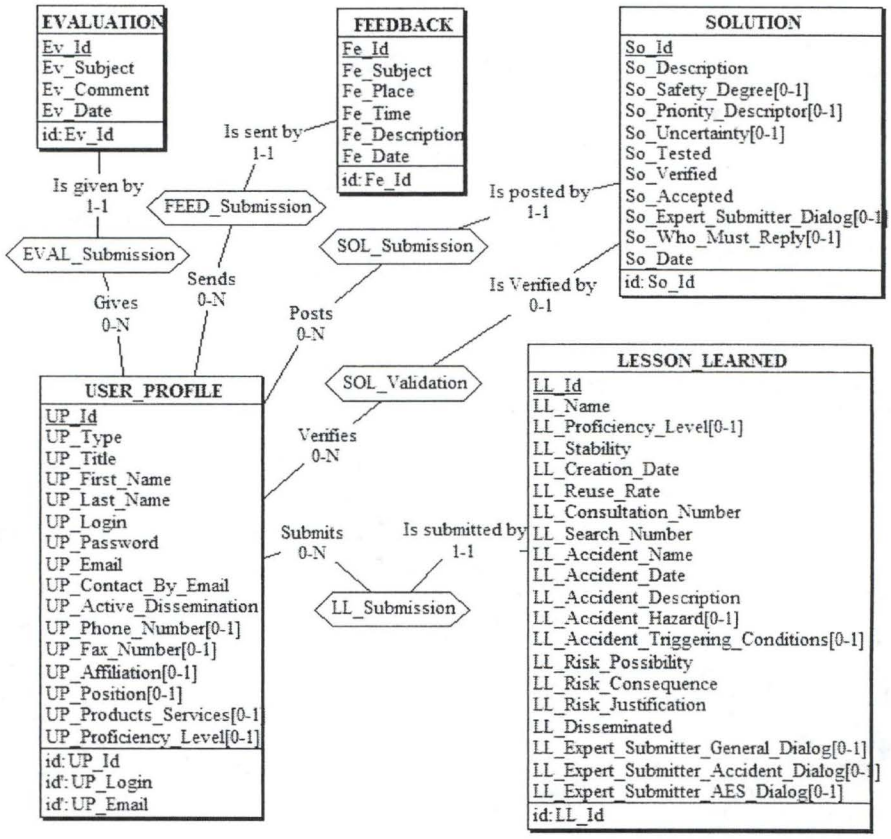

Figure 9.5: E/R subschema (4) - Interaction between user profile and LL / solution / feedback / evaluation.

Chapter 10

Architecture of the lessons learned system and tools used

10.1 Architecture

The architecture of our LL system is a **web-based architecture**.

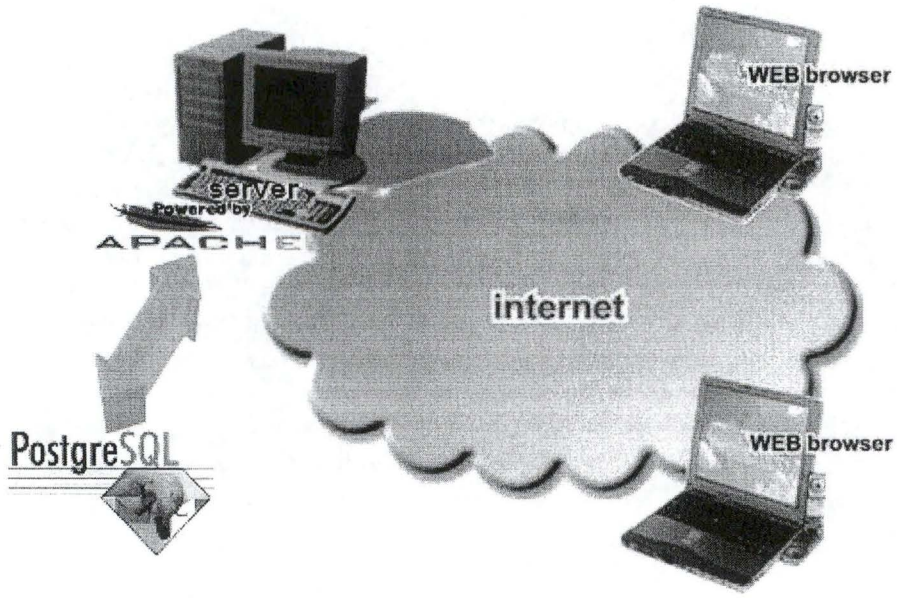

Figure 10.1: Web-based architecture.

There is one server, and users can access the system from everywhere through a web browser.

The server follows a classical **3-tiers** architecture. Data access, business logic, and interface code are separated. It is a Model View Controler (MVC), i.e. there is a clear separation of user-interface-control and data presentation from application-logic. All database accesses are made by specific classes, which encapsulate data access. Interfaces are recorded in a style sheet, in such a way that it is easy to modify it.

10.2 Tools used and technical choices

The only technical constraint imposed was to work with open-source software for obvious cost reasons. Keeping in mind that the final application had to be a prototype, we decided to use a scripting language, **PHP**, coupled with an open-source database system, **PostgreSQL**, distributed under the BSD license, which basically allows any use of the code as long as the credits are maintained.

The main reason for using a scripting language was efficiency. A scripting language is the ideal solution for quick development and prototyping.

10.2.1 PHP and Apache

Considering the convenience of **PHP** versus **Perl** and the nature of our work, we opted for PHP language. PHP is a free server side scripting language. It can be built into web servers like **Apache** and one can use it to generate pages dynamically. Unlike Perl, which is a general purpose scripting language that one can use for a wide variety of purposes (and not just generating web pages), PHP was designed from the ground up for scripting web pages. As a result, it has a large number of built-in facilities. Accessing databases is just as easy. There are built-in facilities in PHP to access PostgreSQL and many more databases. However, PHP is not the perfect solution for all web site needs. It probably cannot beat Perl in terms of convenient and efficient text crunching, but in the case of our LL system prototype, this is not an important factor. Additionally, learning PHP is a piece of cake and we could get started writing our scripts after a very short learning period.

10.2.2 PostgreSQL

We have chosen **PostgreSQL** because this database system is ACID compliant, unlike **MySQL**, and it manages potential concurrent accesses to the database. MySQL is not ACID compliant because it does not support

consistency, isolation, nor durability. However, MySql supports atomicity using table locks. Because of its limited feature set, MySQL is very fast.

PostgreSQL offers many more features and one can be confident that data are safe. PostgreSQL also supports a richer SQL dialect than MySQL: PostgreSQL supports subqueries, stored procedures, views... Furthermore, PostgreSQL's advanced features are more likely to be stable than the newer MySQL equivalents, having been implemented for a longer time. Regarding our need of features like subqueries and the non-priority of fast queries (do not forget that it is a prototype!), we have adopted the PostgreSQL system.

10.2.3 Monitoring with macros

One of the goals of the prototype was to monitor every action of the user and deducing which LL he needs and when (*"the right time at the right place"*), as in the proactive dissemination. Unfortunately, the period of time to develop such a system was too short. Despite the lack of time, we have developed a tool allowing interaction between the LL system and an Office application, as in the reactive dissemination. The easier technology allowing user to adapt an Office application is the macro way.

Chapter 11

Implementation results

This chapter gives an overview of the final application and graphical interfaces. Each screenshot is linked with its corresponding UC. Only the most interesting functionalities are presented in this chapter.

11.1 Navigation menu

The application has a general screen which possesses a navigation menu on the left. Menu items are different depending on whether the current user is logged in or off (figures 11.1 and 11.2), and whether he is a normal user (figure 11.2) or an expert moderator (figure 11.3). The expert menu is the same as the normal user menu. The main difference is that experts can validate new LL and solutions.

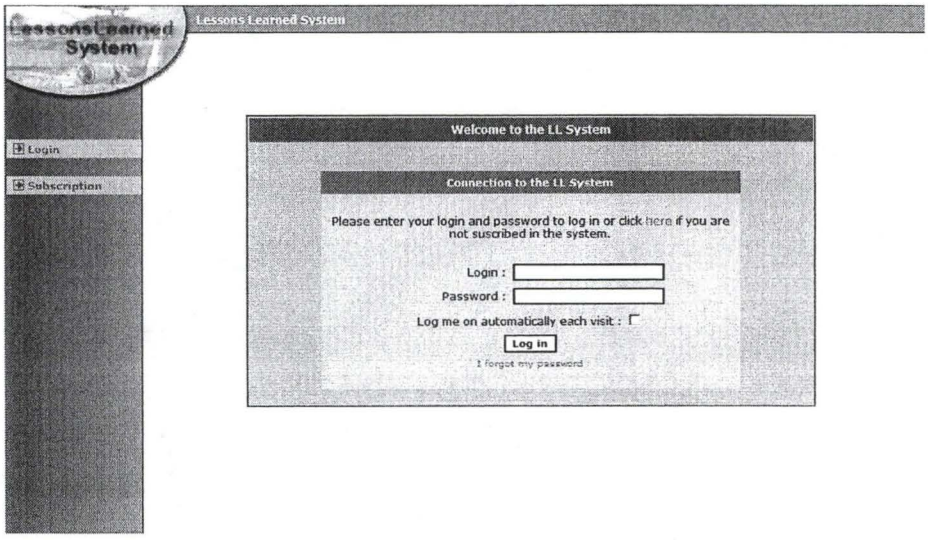

Figure 11.1: Menu when user is logged out.

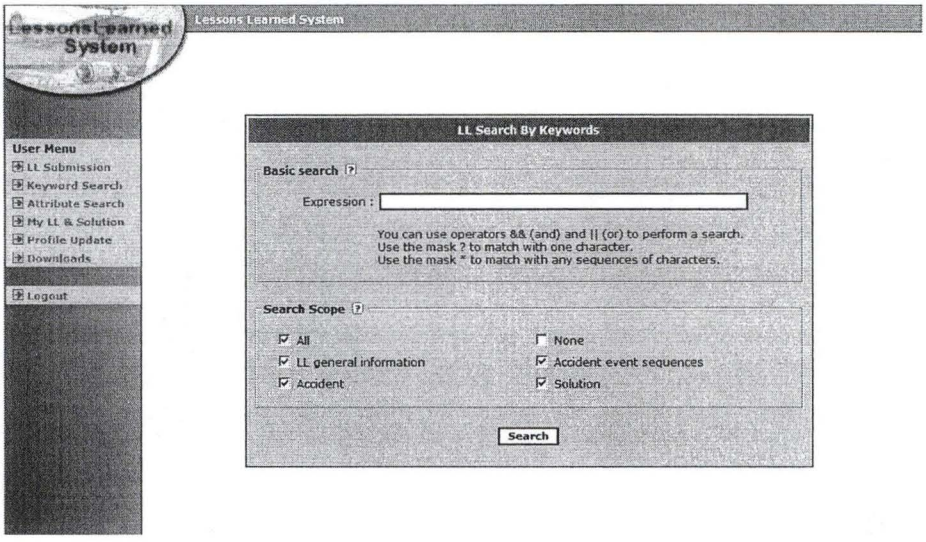

Figure 11.2: Menu when normal user is logged in.

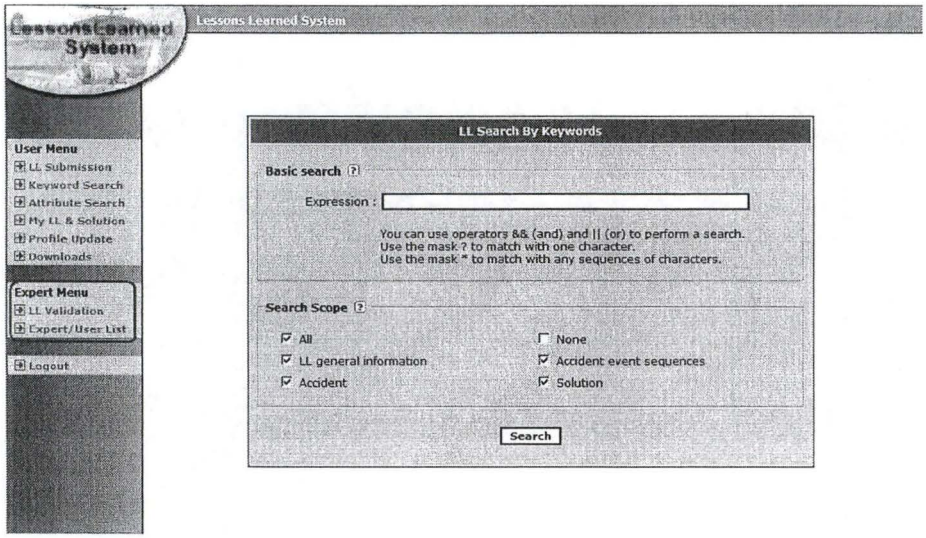

Figure 11.3: Menu when expert is logged in.

11.2 Profile management

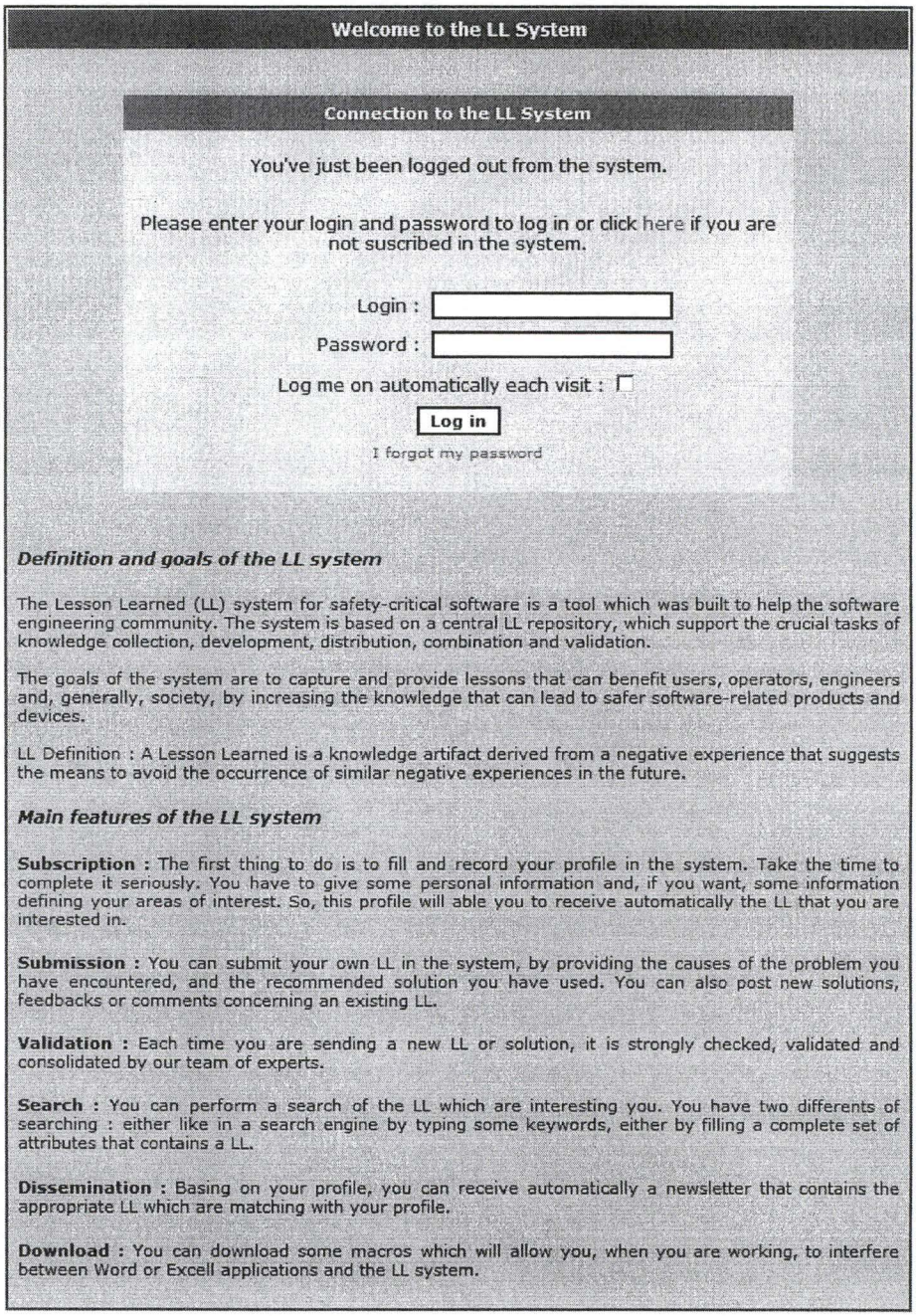

Figure 11.4: Welcome and login.

The first screenshot above on figure 11.4 is the login and welcome screenshot that contains general information about goals and features of the LL system. It corresponds to *UC2 Identification.* If the user has not subscribed, it invites him to fill in a form to be recorded in the LL system. He can ask to receive by mail his password if he forgot it. He can also check an option to be logged automatically the next time he visits the LL system.

The four next screenshots on figures 11.5-11.6-11.7-11.8 are relative to *UC1 Profile subscription.* They show the various steps a user has to go through when filling in his profile:

1. He firstly gives personal and general information.
2. He gives information about LL in which he is interested.
3. All the data entered by the user are displayed and the system asks him to confirm his profile record.

Most of the screenshots have an help icon (figure 11.9). If the user moves on it his mouse pointer, an explanation box appears.

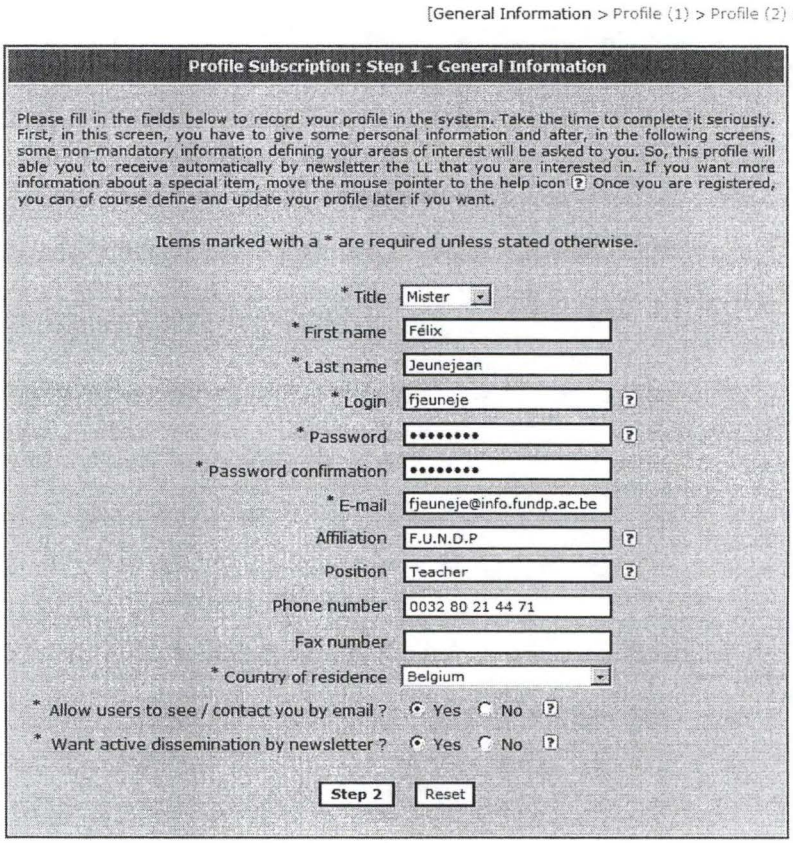

Figure 11.5: User profile subscription (1).

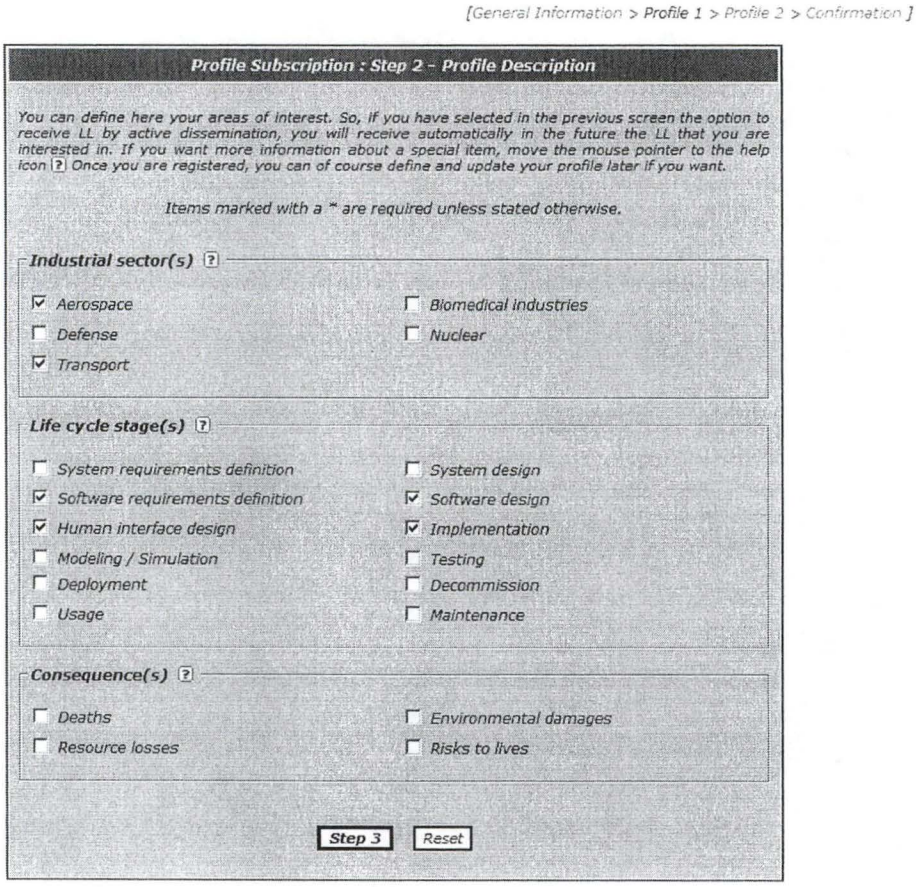

Figure 11.6: User profile subscription (2).

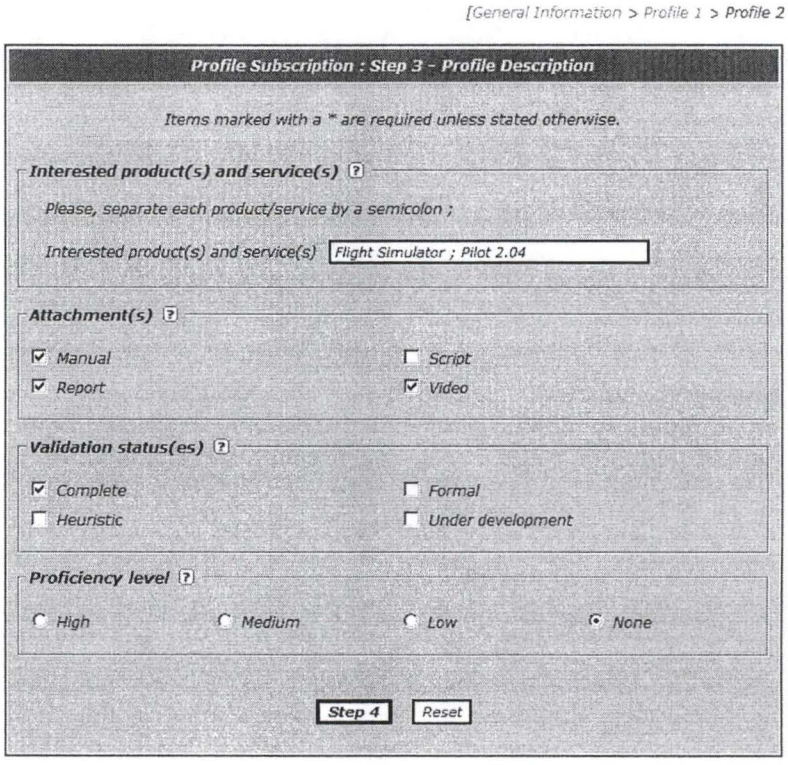

Figure 11.7: User profile subscription (3).

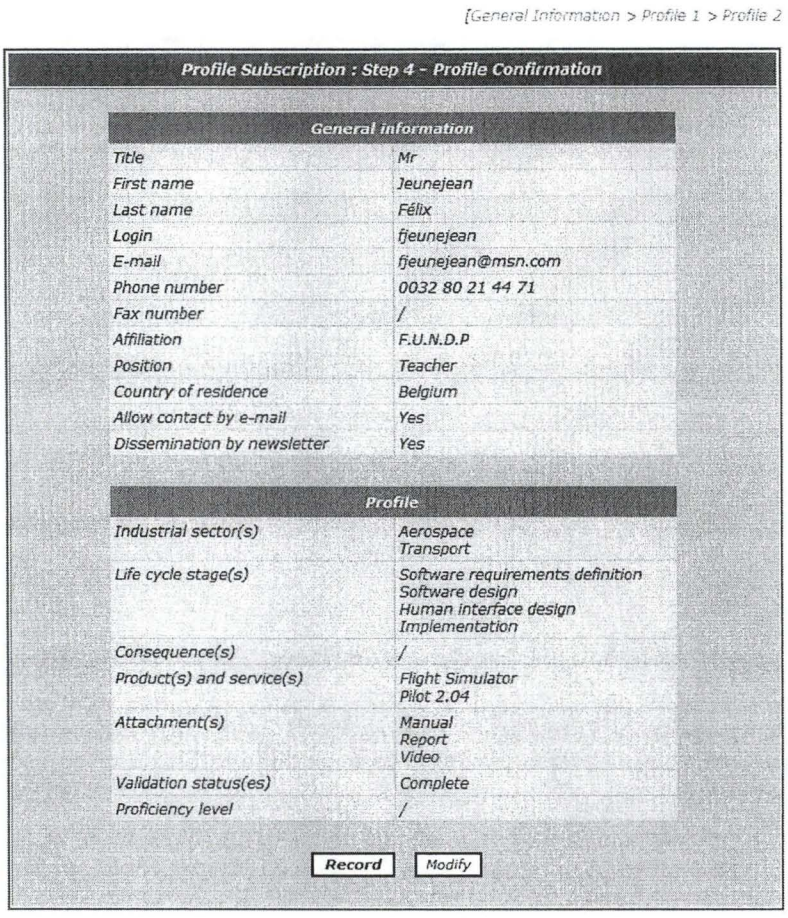

Figure 11.8: User profile subscription (4).

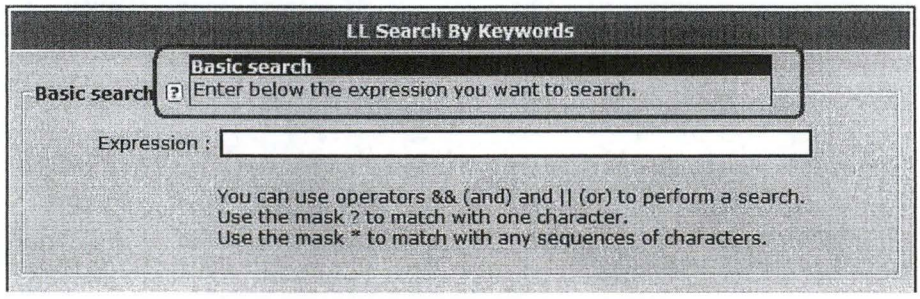

Figure 11.9: User profile subscription - Help icon.

11.3 Collection

The four next screenshots represented on figures 11.10-11.11-11.12-11.13 concern the steps leading to the submission of a LL (general information, accident and event sequence, solution, and record confirmation). They correspond to *UC5 Submit a LL*.

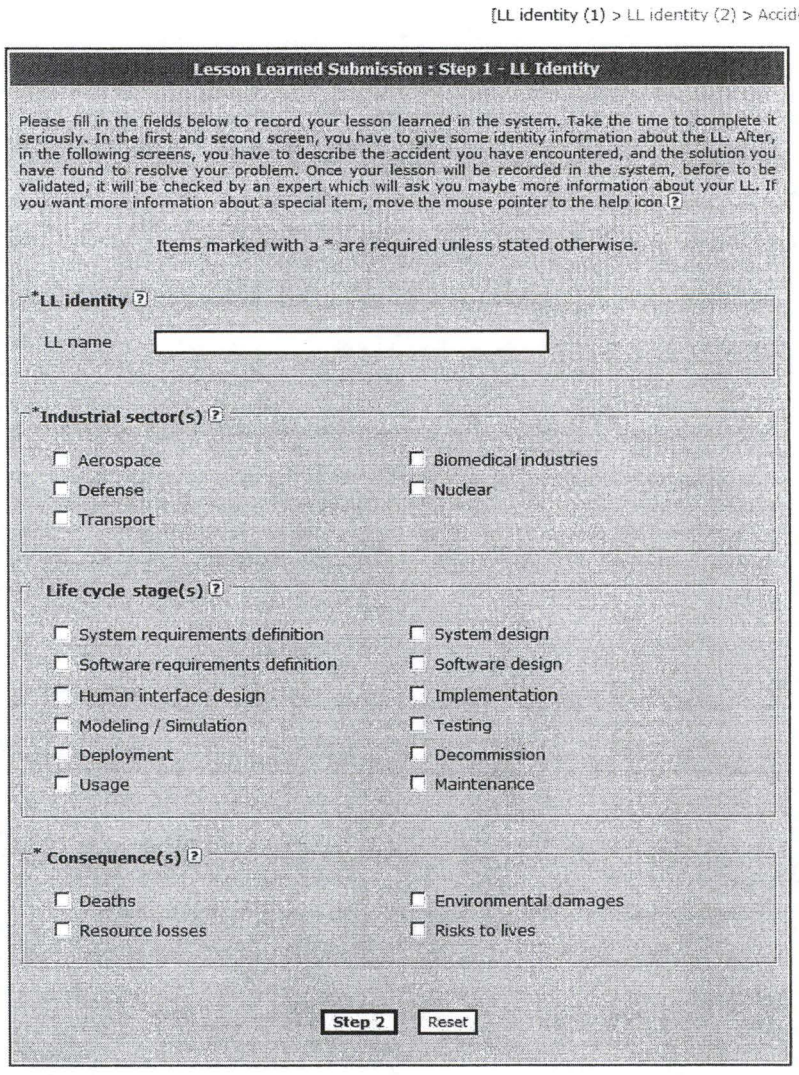

Figure 11.10: LL submission (1).

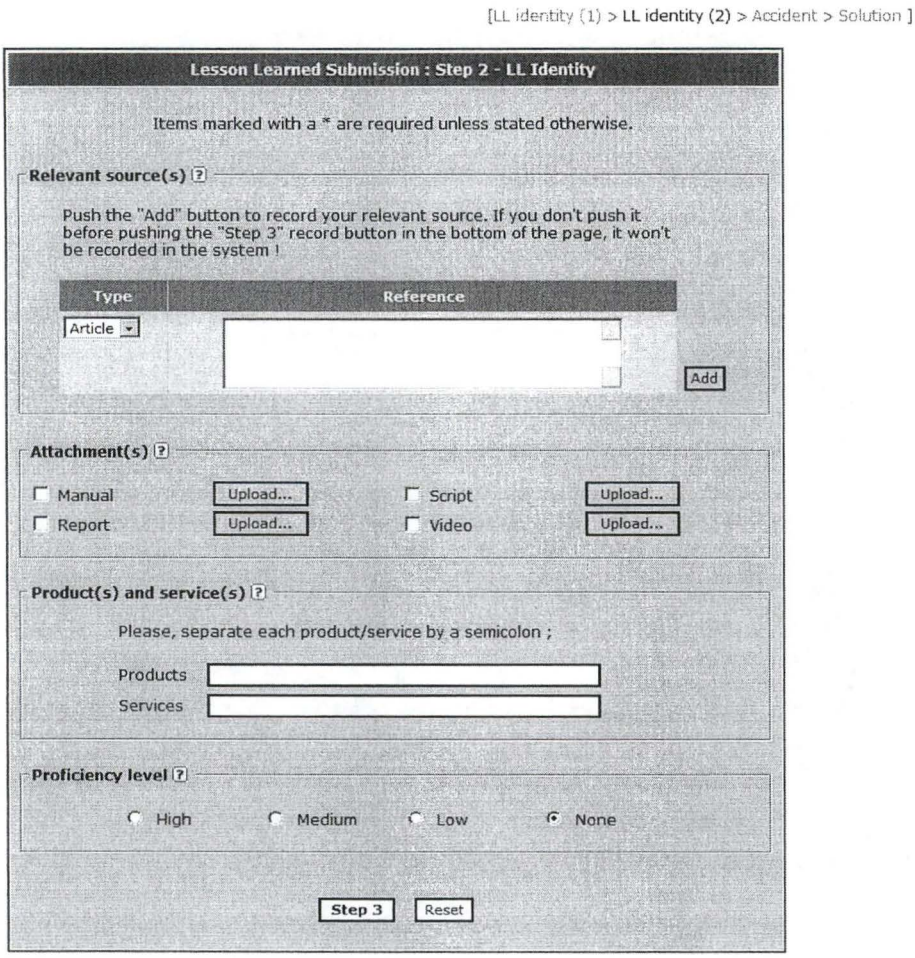

Figure 11.11: LL submission (2).

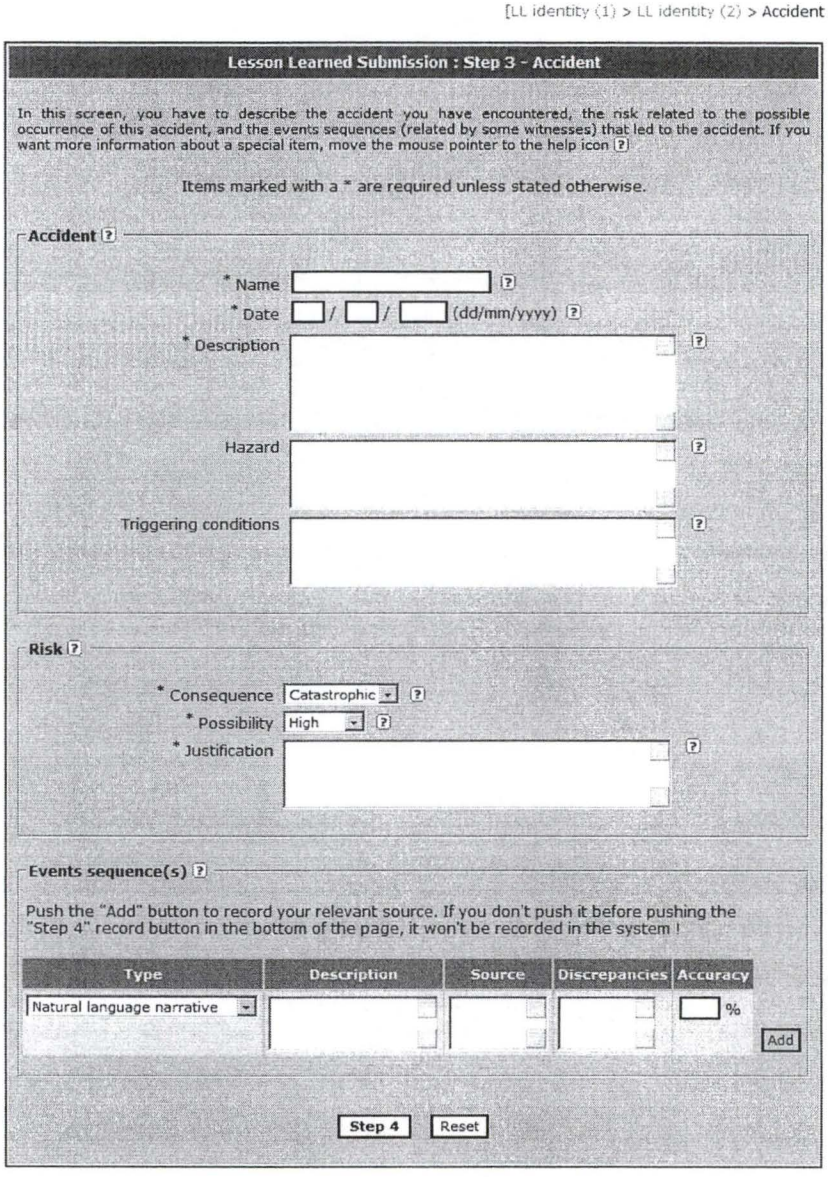

Figure 11.12: LL submission (3).

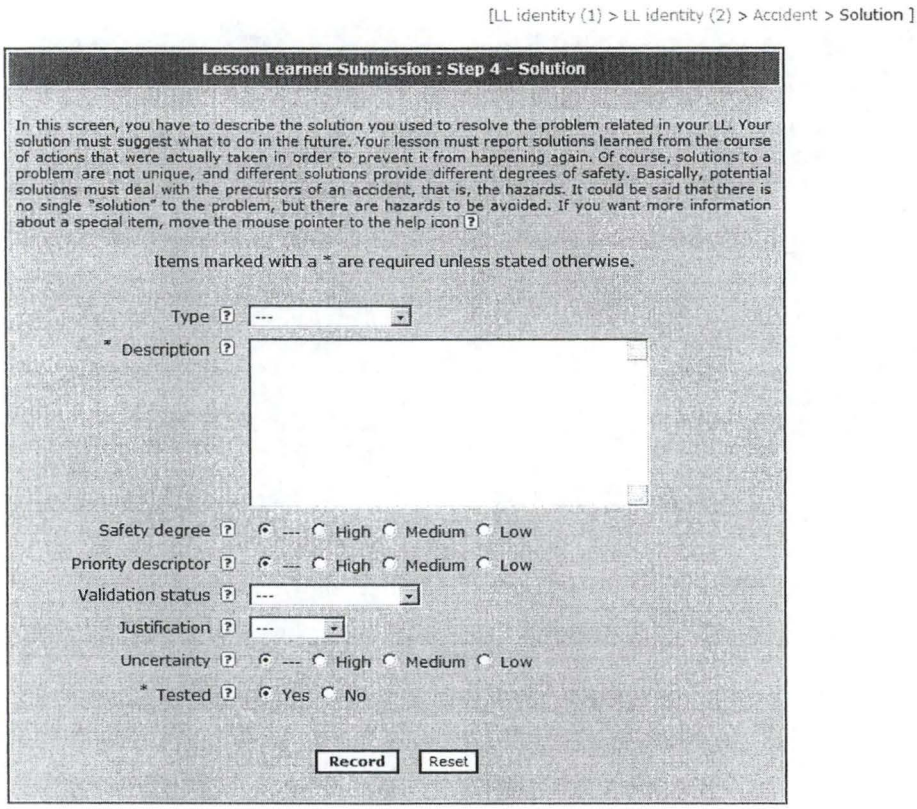

Figure 11.13: LL submission (4).

11.4 Validation

The following screenshot on figure 11.14 is linked with the *UC13 List non verified LL and solutions* that allows the expert to ask for non verified new lessons and solutions related to his area of knowledge. He can see the state (*"needs modification by submitter"* or *"needs validation by expert"*) of each lesson and solution, i.e. if submitters have replied to his questions concerning a specific lesson or solution.

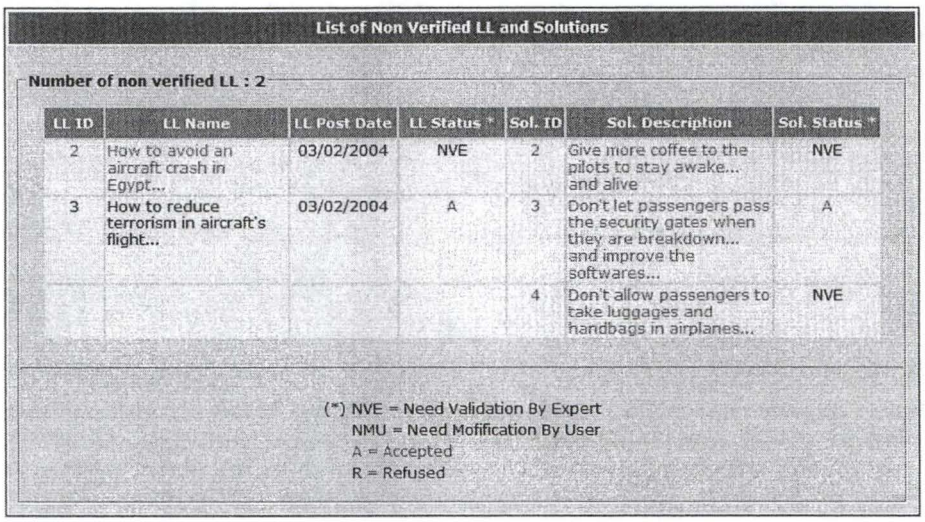

Figure 11.14: Non verified LL and solution retrieval.

The three figures 11.15-11.16-11.17 below are related to *UC10 Update a LL*, *UC11 Validate a LL*, and *UC12 Validate a solution*. After having selected a new non verified LL or a new non verified solution on an existing LL (see figure 11.14), the appropriate expert can update the LL/solution and post some comments to ask the submitter more information by email. He can add links to related existing LL. Consequently, a complete dialog by mail and throughout the LL system happens between the expert and the submitter. Once all is clear for each of them, the expert decides to accept or reject the new LL/solution. If he rejects it, he should explain the reasons that lead to this choice. The submitter is then notified of the expert decision.

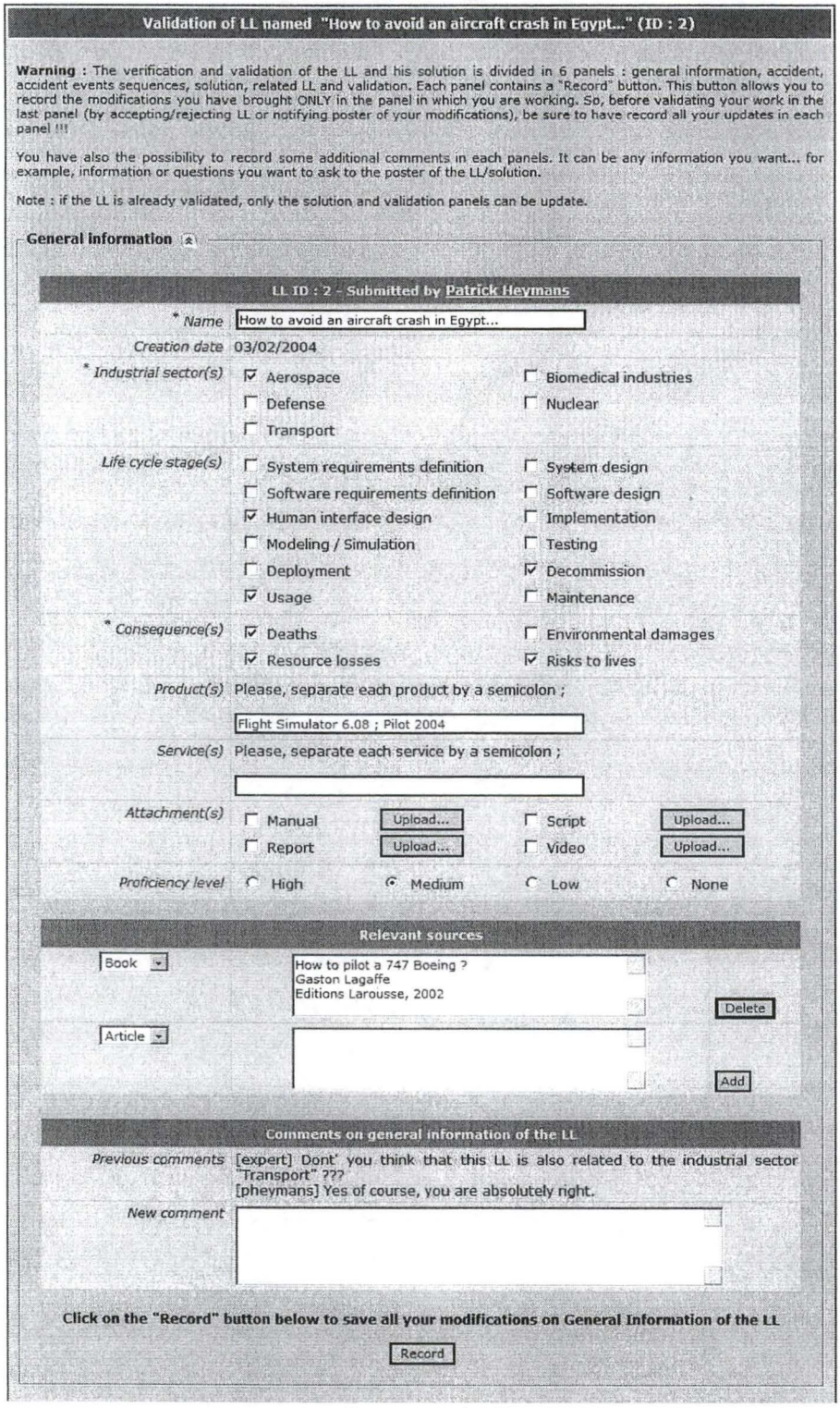

Figure 11.15: LL validation (1).

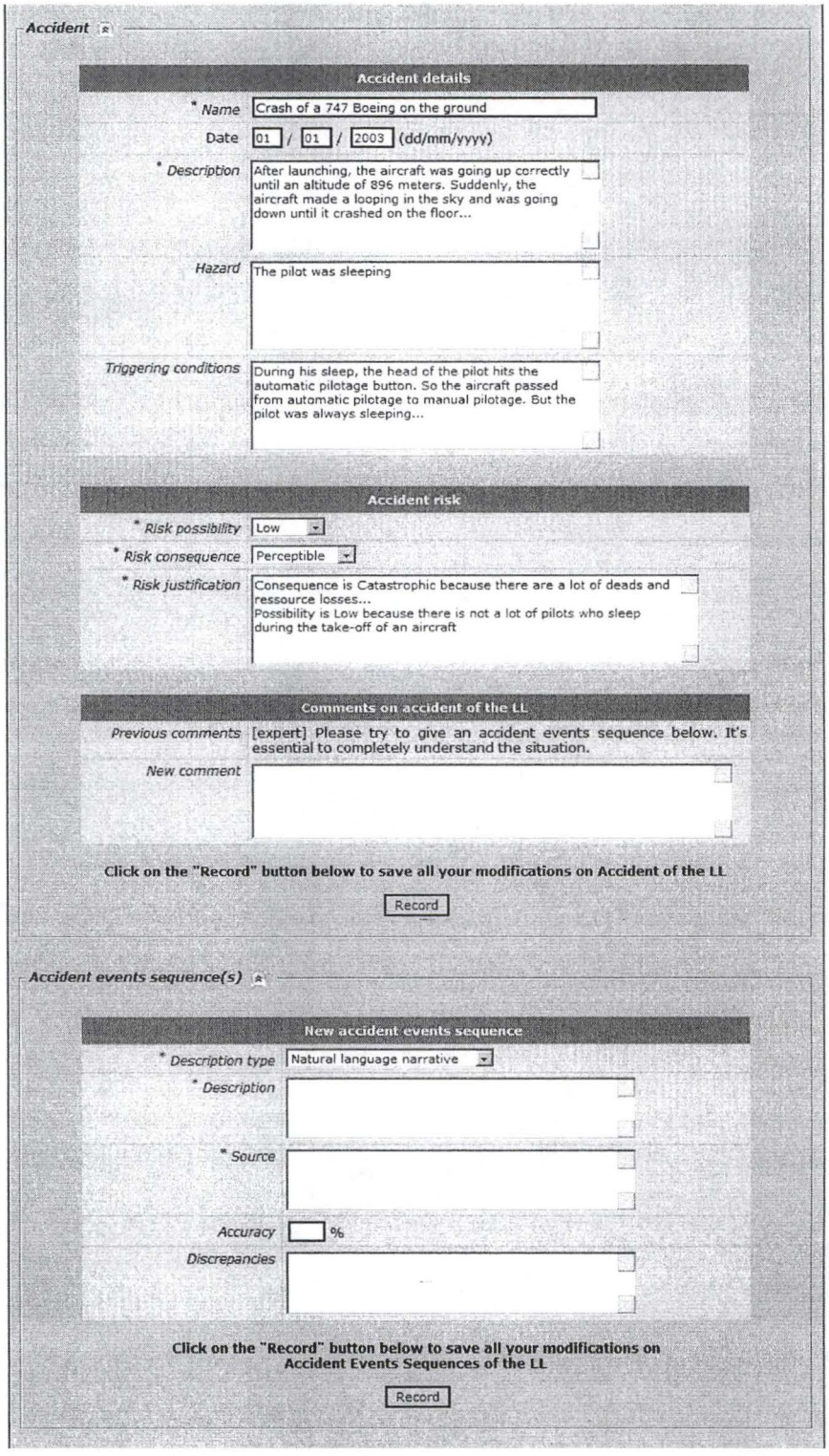

Figure 11.16: LL validation (2).

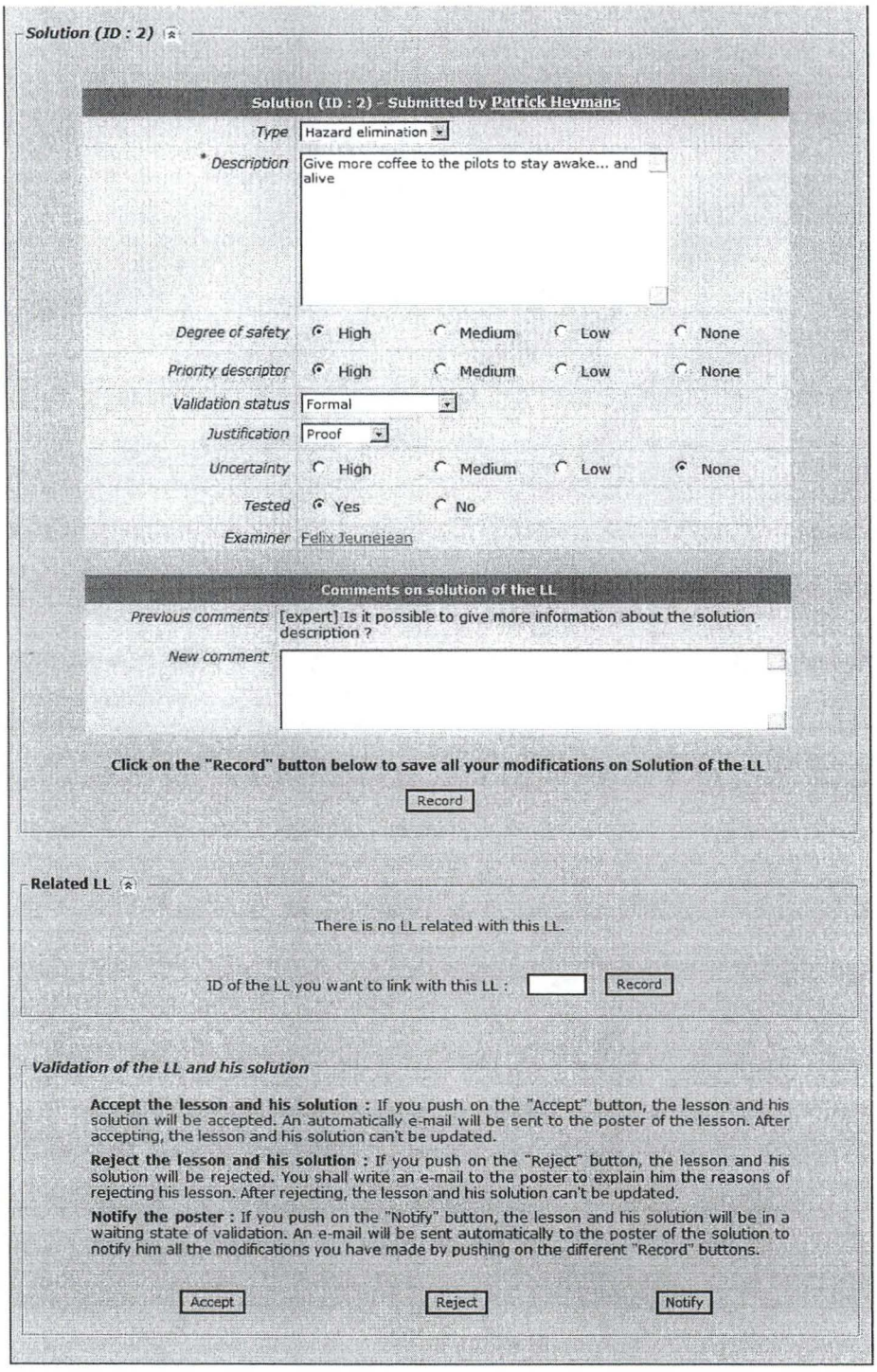

Figure 11.17: LL validation (3).

The last screenshot related to the validation process, represented on figure 11.18, is connected with *UC15 List my submitted LL and solutions* and *UC14 Reply to LL expert's questions.* The submitter asks to see all verified or non verified LL and solutions he posted. He can see the state (*"accepted"*, *"refused"*, *"needs modification by submitter"*, *"needs validation by expert"*) of each of them. If he clicks on one of his submitted LL or solutions, this one is displayed and he can reply to the questions asked by the expert, if any.

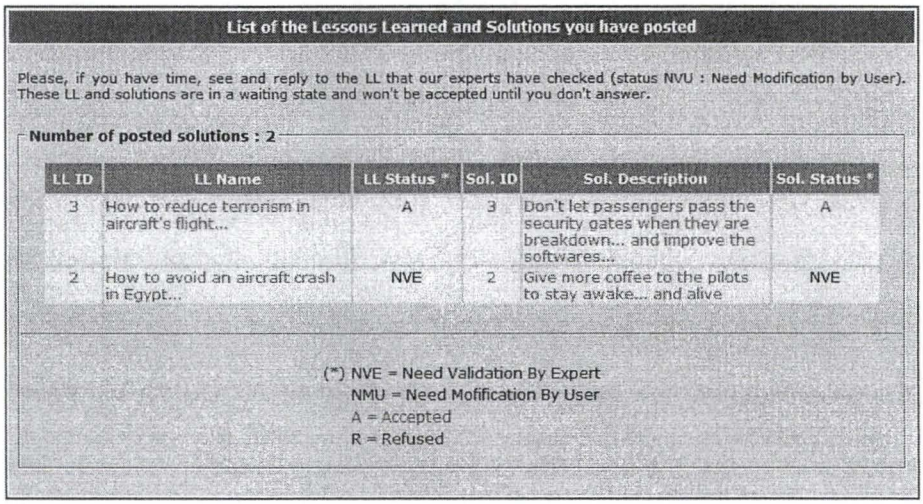

Figure 11.18: My LL and solutions.

11.5 Dissemination

Figures 11.19-11.20-11.21 are related to passive dissemination. The first screenshot on figure 11.19 concerns *UC17 Search by keywords*, where users can specify the scope of their search and use search operators between keywords.

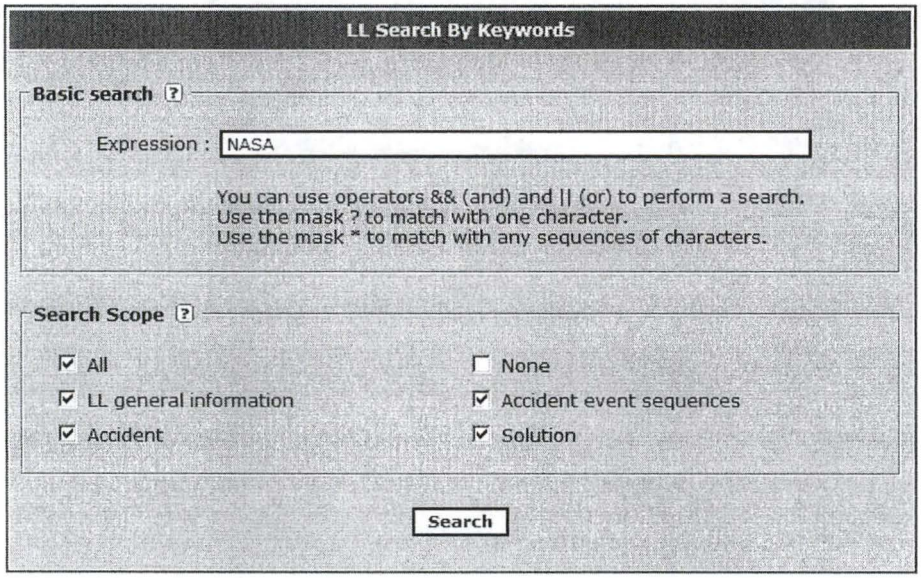

Figure 11.19: LL search by keywords.

The second screenshot on figure 11.20 concerns *UC16 Search by attributes*, where users search for LL by giving values related to fixed LL attributes. They can also specify their search with additional keywords and precise the scope of their search. The *scope* of the *"and/or"* list boxes concerns the matching of different values in a specific attribute. For example, concerning the industrial sector attribute, users can make a search with *"Biomedical industries"* and *"Transport"* attributes, by using the CTRL key. The *matching rules* concern matching among several attributes and determine if at least one attribute must match or all attributes must match.

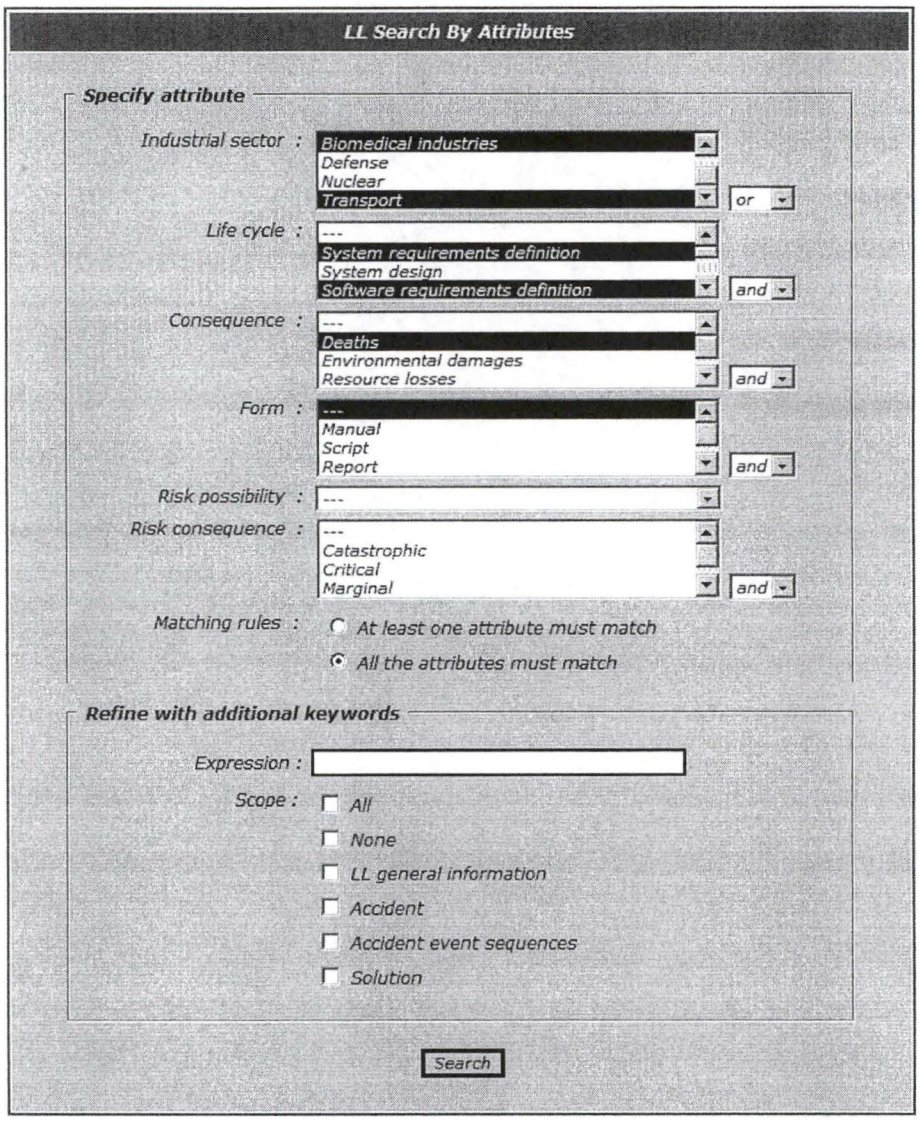

Figure 11.20: LL search by attributes.

Figure 11.21 shows the search results. For each result, the user can directly download the PDF document corresponding to the LL. The icon next to the PDF icon shows a more complete description (other attributes) of the LL found. The user can also refine his search by performing a search by keywords only in the LL found.

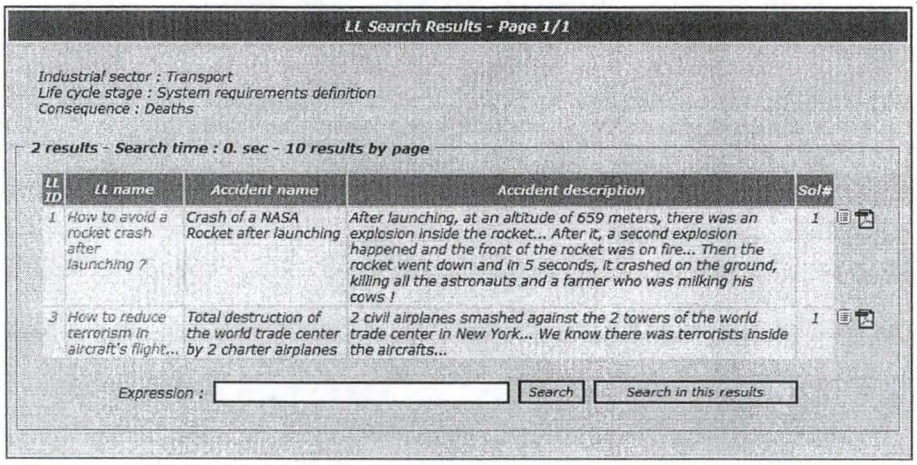

Figure 11.21: Search results.

Figures 11.22-11.23-11.24-11.25 concern the display of a LL, once the user has performed a search (passive dissemination) and selected this LL (see figures 11.19-11.20-11.21). They relate to *UC9.1 LL selection and display.* The display of a LL contains various panels that the user can open or close: general information about the LL (identification, relevant sources and statistics), accident description and risk, accident event sequence(s), solution(s) with eventual feedback(s) and evaluation(s), and related LL.

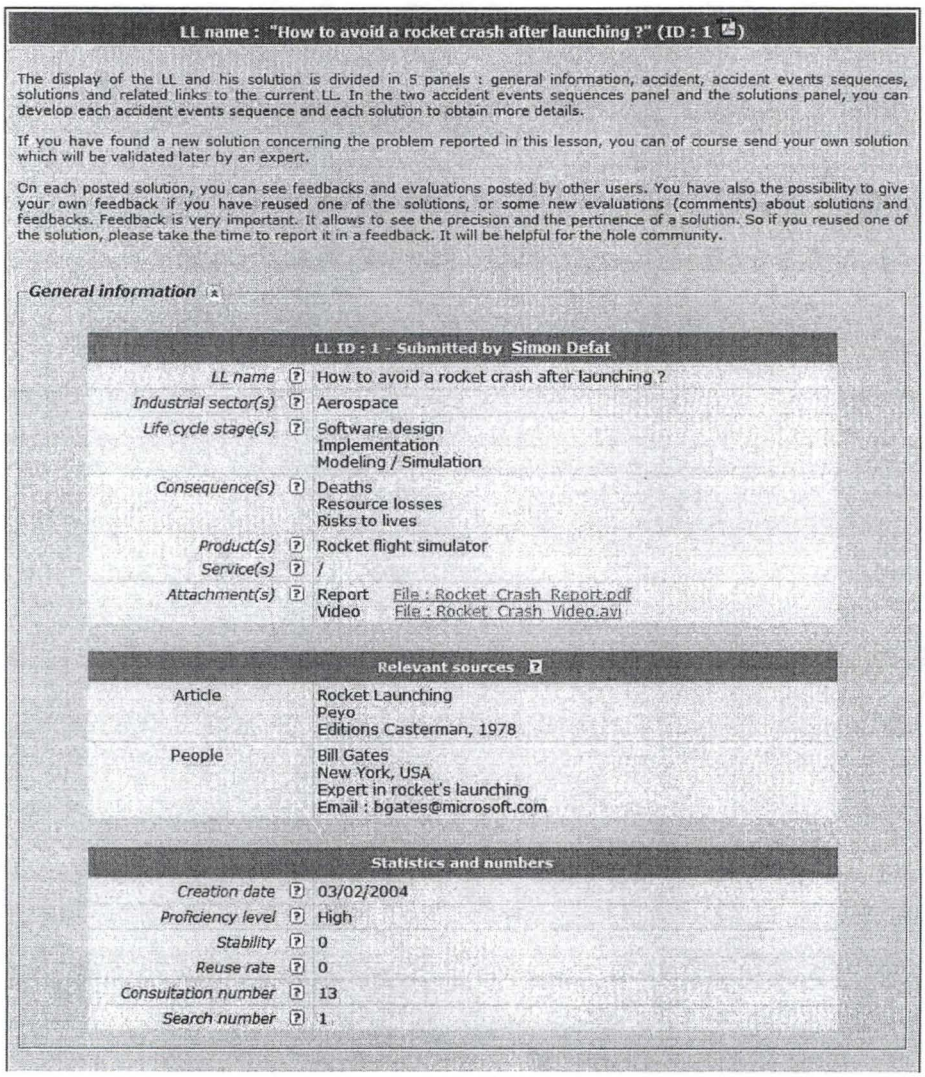

Figure 11.22: LL display (1).

When the user consults a LL, he has different options. He can download it in PDF format. He can post a new solution, or a new feedback/evaluation on an existing solution. If the submitter has given his permission, the user can directly contact him by e-mail to ask more information. He can also contact the expert who checked the LL.

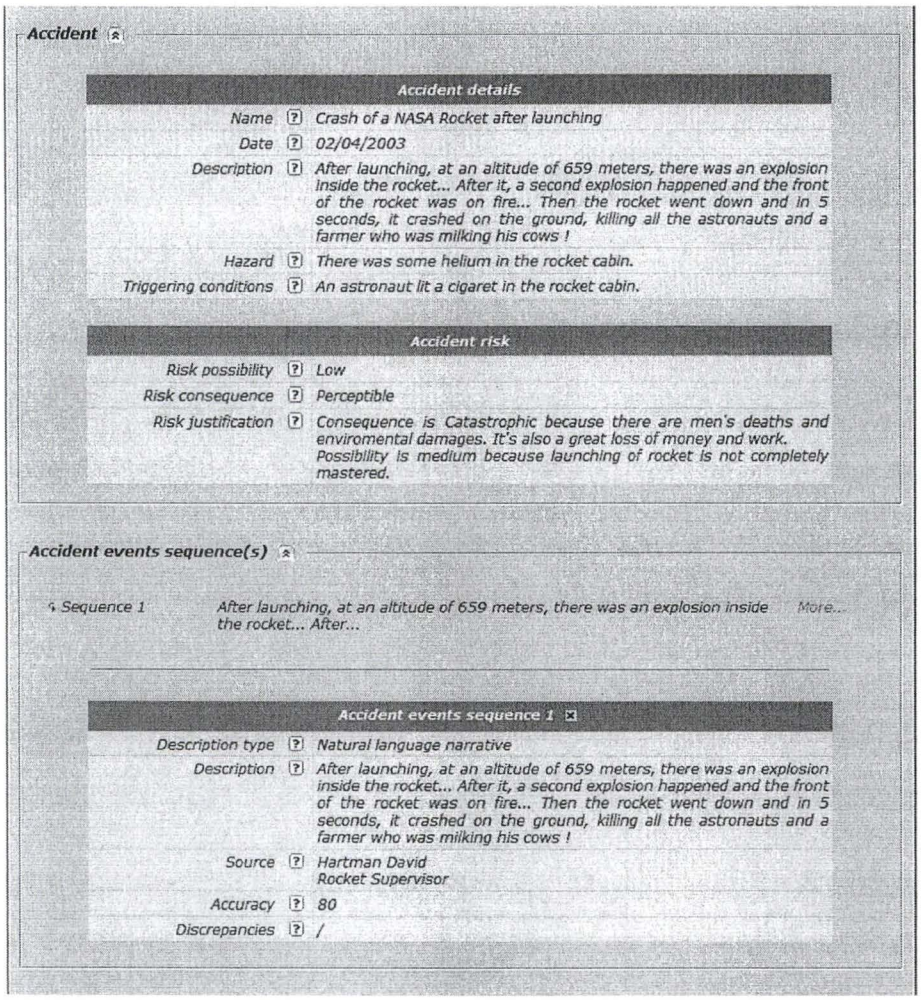

Figure 11.23: LL display (2).

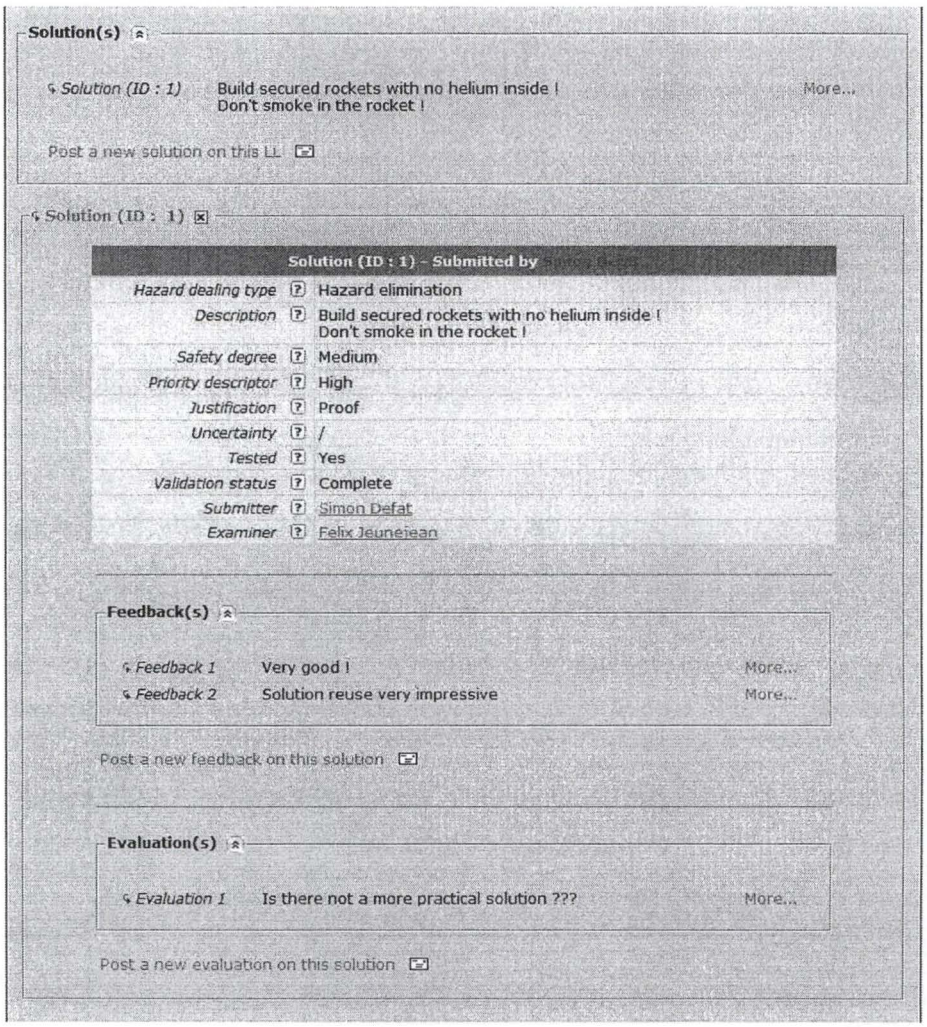

Figure 11.24: LL display (3).

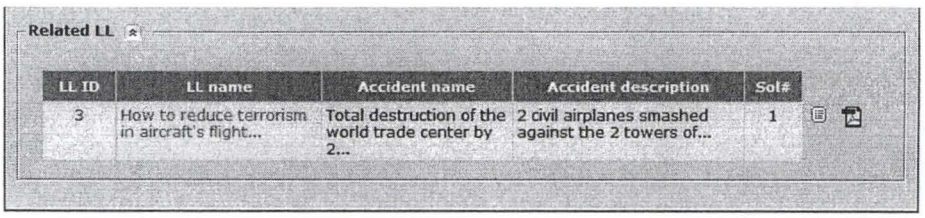

Figure 11.25: LL display (4).

The two next screenshots on figures 11.26-11.27 are related to *UC18 Macro download* and *UC19 Search with macro call*, that allow users to make external call and search for LL, through Microsoft Word or Excel applications.

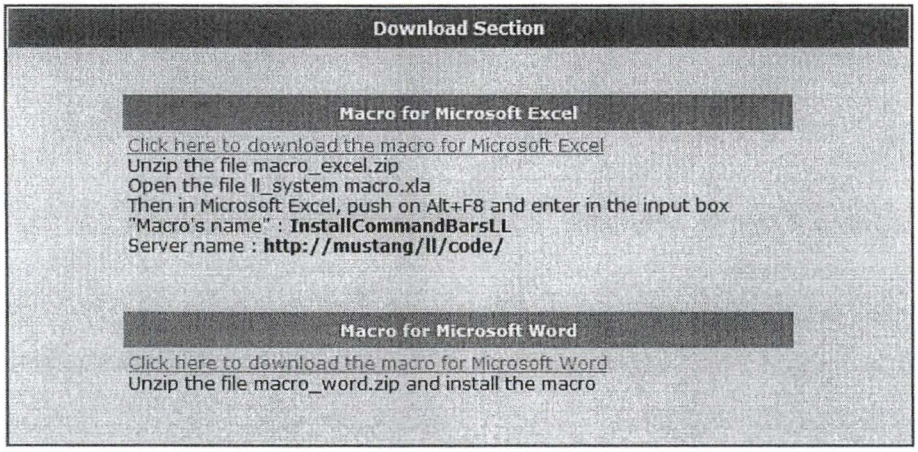

Figure 11.26: Macro download.

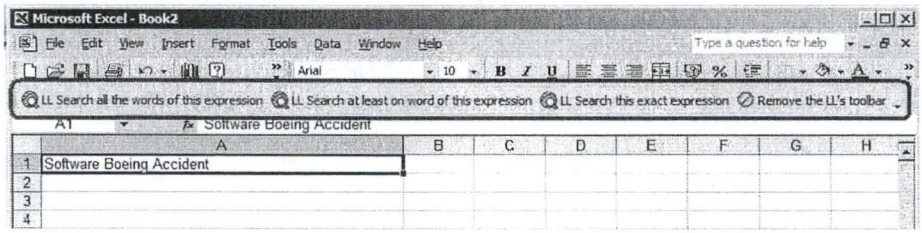

Figure 11.27: Macro call.

There is no screenshot that corresponds to *UC20 Start active dissemination*, because each time a LL or solution is validated by expert, this is automatically disseminated by email (newsletter) to users whose profiles match.

11.7 Administration

The screenshot on figure 11.29 corresponds to *UC21 Manage experts* where the administrator gives/removes rights to a normal user/expert to become an expert/normal user.

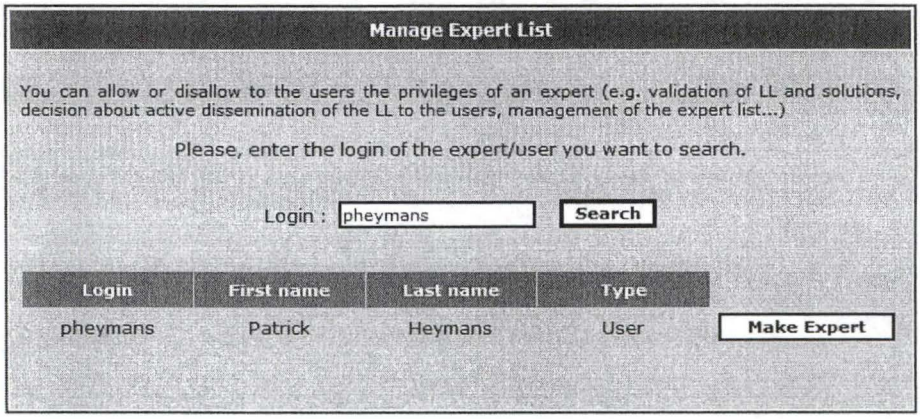

Figure 11.29: Expert management.

11.6 Reuse

The screenshot represented on figure 11.28 is related to *UC9 Feedback selection and display* and *UC8 Submit an evaluation*, where the user submits a comment on a feedback that has been posted on a solution. The functioning of *UC7 Submit a feedback* and *UC9 Evaluation selection and display* is the same.

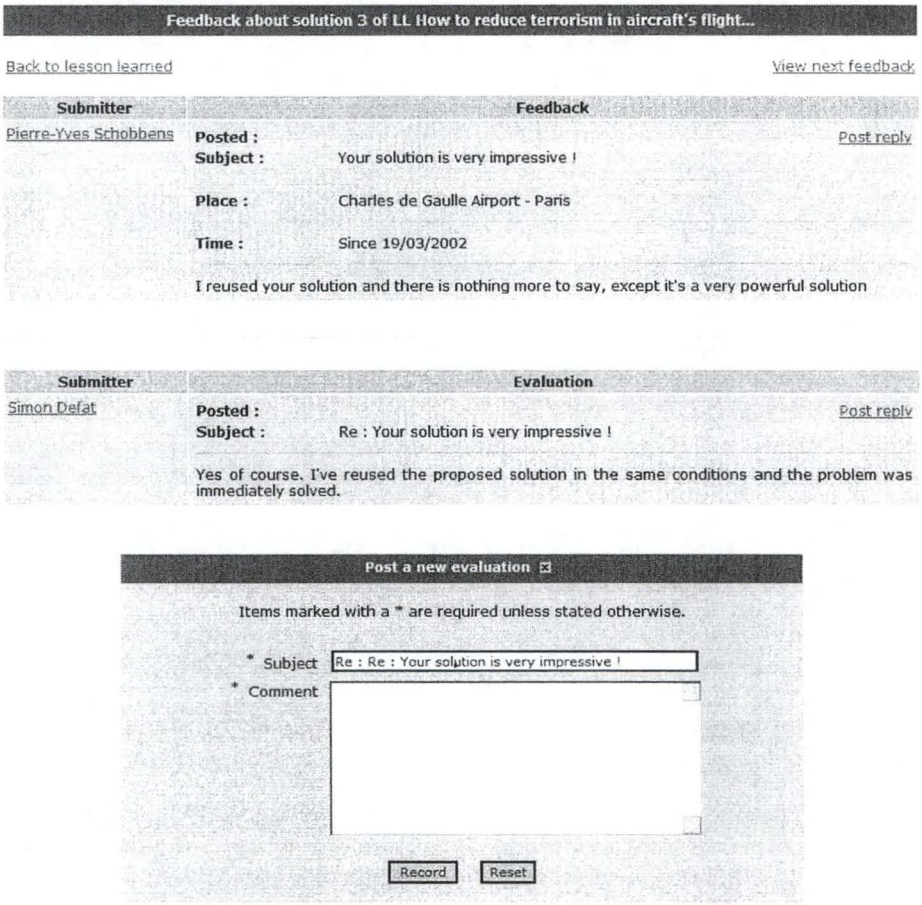

Figure 11.28: Feedback selection and evaluation submission.

Figure 11.30 is linked with *UC22 Manage database* where the administrator adds or removes new values related to fixed attributes in the database, as industrial sector or life cycle stage values.

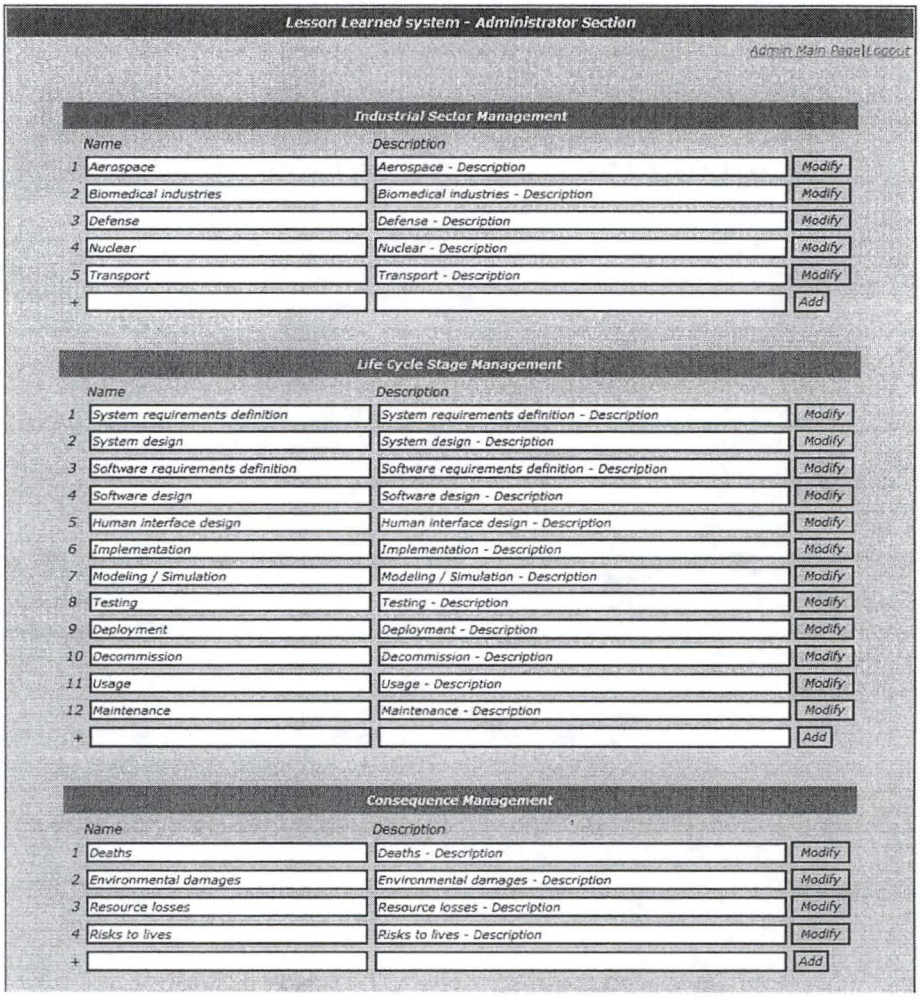

Figure 11.30: Database management.

Part III

Creation of an environment for lessons learned through knowledge management

Chapter 12

Organizational culture and cultural barriers

This chapter and the following ones in this part are based on [**GAO, 2002**] and [**Rus & al., 2002**]. They suggest solutions to organizational and human problems caused **after** the development and setting up of LL systems and, more generally, KM systems.

Despite the procedures and the processes established to capture and share LL, it is not certain that lessons are being reused and applied in the future. A lot of organizations, as the NASA for example, have noted that LL are not routinely identified, captured and reused by projects managers and programmers or, more generally, employees.

The first main reason is that there exist **cultural barriers** and **resistance** to the sharing of LL. The second reason is that a LL system is important but should not be the **only mechanism** to share knowledge.

12.1 Individualism

Although new technology can be very helpful in sharing knowledge, organizational culture might not support it. Some cultures promote individualism and ban cooperative work. It is very frequent that workers do not want to give away their knowledge, or reuse someone else's knowledge. Some think that software engineers are indisposed to reuse solutions found by other employees. If organizations do not encourage a **knowledge-sharing culture**, employees might feel possessive about their knowledge and will not share it.

Lack of a knowledge culture is the main reason why KM strategies failed. It was proved that a large number of organizations failed because

they did not establish their goals and strategy before implementing LL systems. Employees should be convinced of the system utility and importance.

However, the obstacles might not be as big when it comes to safety-critical systems since *"There is a tendency within any industrial sector to collaborate in relation to safety issues, as negative incidents can affect the whole sector"* [Leveson, 1995].

12.2 Lack of trust

A big problem is that a large number of employees are not persuaded of the effectiveness of LL systems. They think that no benefit can be drawn from LL.

This type of behavior and individualism can cause the ruin of a LL or other types of KM systems. Employees are conscious that organizations appreciate them for their own personal and specific knowledge; they might be afraid that they will be judged as superfluous or useless once organizations have caught their knowledge.

12.3 Intolerance for mistakes

Employees might not be willing to share negative experiences and LL based on failures because of their **negative connotation**. They have a perception of intolerance for mistakes. Consequently, although the main goal is to prevent the same errors, employees might fear that such information could be used against them.

12.4 Lack of time to share knowledge

Another problem is that employees do not often have time to input or search for knowledge. They should be stimulated to take time to compose and submit their LL in the repository. This constraint is often hard to accomplish.

Although adaptation is difficult, such behaviors should be reviewed and replaced by a constructive approach that promotes and rewards sharing.

Chapter 13

Recommendations to incite sharing knowledge and using lessons learned systems

13.1 Reward systems and performance evaluation

A good idea to incite workers to share their knowledge by submitting LL or to search and reuse existing LL, is to settle a reward system. Such a system was set up by **Xerox**. It was suggested to establish a **"hall of fame"** for employees whose contributions would solve real business problems. Xerox LL contains also the identity of the submitter, that could reveal his **reputation** among the company. Consequently, it is an important incentive to bring his involvement by using the LL system.

[Powers, 1999] says that Xerox created an authentic knowledge-sharing culture through its knowledge repository, called Eureka, which is used by more than 25,000 employees worldwide. Xerox's Eureka LL system is powerful because it is totally filled by its own users; no specific employee is in charge of creating and submitting LL. Eureka allows Xerox to save between five and ten percent on work costs. Knowledge sharing is part of the employees' daily work.

Scientists at **NASA's Langley Research Center** are **monetarily rewarded** if knowledge they capture and share is reused.

Ford recompenses workers who send LL used within the organization. Managers are also motivated to share because they are **evaluated annually** on the basis of knowledge sharing. Similarly, employees' performance of the **World Bank** is estimated in the same way.

Other examples of reward systems can be mentioned. Bruce Karney, evangelist of a **Hewlett-Packard** KM initiative, gave out **free Lotus Notes licenses** and **free airline miles** to prospective users. **Infosys** rewards employee contribution and use of knowledge with **knowledge currency units**, which they can convert into cash. The online expertise provider **Expert Exchange** rewards experts with **points** for answering questions and recognizes those with the most points on the front page of their web site.

13.2 Additional mechanisms for lesson learning

Programmers and project managers are directed to review and apply LL, but a LL system alone is not sufficient in an organization. In section 2.4, we explained that there were two forms of knowledge: tacit or explicit. LL system insures that it can collect explicit as well as tacit knowledge, but actually, tacit knowledge can not always be gathered in such a system. Hence, there should be additional mechanisms that promote a deep KM culture throughout the organizations.

13.2.1 Mentoring

Ericsson creates a new concept to exchange tacit knowledge, instead of storing it in a repository. Two roles have been defined to spread tacit knowledge to a larger number of employees:

1. The **experience communicator** is an employee who is specialist in a particular field.
2. The **experience broker** links the experience communicator with the **employee** who faces a problem.

The communicator should not solve the problem himself but instead coach and instruct the employee on how to solve it.

13.2.2 Storytelling

Another example is storytelling. The **NASA** encourages senior programmers to tell personal experiences and to disseminate their knowledge through a series of **short stories** available on **their web site**. Each story deals with a topic that will help other programmers to succeed. They can also provide with online training resources, such as project management tools, that can be reused and help others.

13.2.3 Other means

Supporting lesson learning can also be done by designing **training programs**, **pair programming**, **job rotation**, **technical reviews**, or **after-action reviews** as in the US Army. All these activities allow experts and employees to exchange part of their knowledge.

Communities of practice established by **Ford** or the **World Bank** is another example of such means. They consist of groups of employees who are bound together by shared expertise and a common passion for joint enterprise [Wenger & al., 2000].

13.3 Strategic plan for KM and knowledge manager

In larger organizations, creating a **team** and designating a **knowledge manager** to coordinate KM activities is essential. It is recommended to position the interaction between lesson learning and KM through a **strategic deployment plan for KM**. This concerns translating the fuzzy notion of KM into a concrete and collective vision in the long term, with fixed objectives for sharing KM within the organization.

Nominating a LL manager is necessary to accomplish, organize and coordinate all lesson learning activities. This manager is also responsible to ensure that the LL repository is maintained and accessible. In large organizations, this task is crucial because lesson learning activities are scattered around various companies.

For example, at **Ford Motor Company**, the chief executive officer plays an important role for knowledge sharing, personally writing **emails** every week to employees with **comments on past week's experiences**. If employees see that using the LL system is important for their boss, it could become important for them.

13.4 Filling up the LL repository

A problem with LL systems is that it might take time before measurable benefits appear. It usually lasts a long time before knowledge bases contain a **critical mass** of knowledge.

In the last part about future works, a possible solution to fill a LL repository, in order to reach a sufficient base of LL, is proposed. It shows that interaction and knowledge exchange between different KM systems (as interaction with comp.risks or distribution and interaction throughout brokers) could help to gather a critical mass of LL.

13.5 Performance measurement

It is important to track and report on the **efficiency** of LL systems and all KM activities, by using objective performance metrics for example.

13.6 Investing in knowledge sharing

Implementing all these recommendations costs a lot and needs investments. Several LL system initiatives fail because financial investments in KM were insufficient. It is observed that successful LL systems are those in which companies spend significant investments, as in the **World Bank** or **Ford Motor Company**.

Part IV

Ethical questions

Chapter 14

Ethical questions

According to [Dumas & al., 2003], LL systems based on KM consist of a series of procedures based on information manipulation, which include information gathering, structuring, analysis, evaluation, recording, retrieval, and accessibility. All these steps imply the notion of qualitative rather than quantitative evaluation and subsequently are liable of ethical consideration. In this chapter, we introduce a short reflection about the responsibility, the privacy statement and the usage of such systems.

14.1 Responsibility and usage problems

LL systems raise the problem of the information responsibility and usage, especially in the field of safety-critical software. If something wrong happens as a consequence of the usage of a LL, who is responsible? Is the responsibility shared among experts, users and LL submitter? Is the expert responsible or only the submitter of the LL? Each answer depends on the legislation of the country where the system is hosted. In order to promote usage of the LL system and to gain the confidence of users and organizations, a charter can be written. As a result, it can really prevent malicious users whose aim is to introduce wrong information in the system in order to cause disasters or damage to other organizations.

14.2 Quality of information and confidence in the system

LL systems cannot be separated from the quality of information and the trust that can be attached to any piece of information, particularly about software-oriented accidents. In order to ensure the best quality of information and to gain the user's confidence, the system can adhere to a privacy

seal program, such as **TRUSTe** organization[1] for example. This is an independent and non profit privacy organization whose mission is to build user's trust on the Internet. It is also preferable fort users' and experts' identities to be certified by an independent organization.

14.3 IEEE and ACM code of ethics

A LL system with an active dissemination mechanism based on monitoring, bringing the information automatically to the user, can raise some questions about the respect of his privacy and the intrusion of the system.

However, the first article of the **IEEE**[2] **code of ethics** and the point 1.1 of the **ACM code of ethics and professional conduct**[3] says that *"all computing professionals must accept the responsibility in making engineering decisions consistent with the safety, health and welfare of the public, and to disclose promptly factors that might endanger the public or the environment"*. This code favors one ethic because it is aimed at making safer systems.

14.4 Bowie and Duska's four questions

While the law is not yet established, deontological principles may be in conflict. In this case, we can adopt an explicit ethics of data usage and transparency, more or less linked to **Scip**[4].

Generally, for all aforementioned ethical considerations, we can apply Bowie and Duska's [Bowie & al., 1990] four questions, which are:

- Is the action good for the user of the LL system?
- Is the action good for the company or the organization?
- Is the action good for everyone affected by it?
- Is the action fair and just?

According to [Bowie & al., 1990], answering *"yes"* to all four questions leads to an ethical action.

[1] http://www.truste.org
[2] http://www.ieee.org/about/whatis/code.html
[3] http://www.acm.org/constitution/code.html
[4] http://www.scip.org/ci/ethics.asp

Regarding this chapter, developers and users of LL systems should take ethical dimension into account when they develop or use such systems. We cannot ignore it.

Part V

Future works

Chapter 15

Interaction between LL systems

This chapter and the following ones explain the different ways and perspectives of research that could have been explored if we had more time to develop a LL system oriented towards safety-critical software. We remind the reader that the main goal of our work is to develop a first exploratory prototype giving an overview of a LL system oriented towards safety-critical software. We describe in these chapters what could be done or changed to improve the tool or build a new one.

Before explaining what the possible interaction between LL systems is, the following table shows the modifications affecting the characteristics of our LL system (described in section 4.3) if **all changes** described in this part about future works were added to the current features of our prototype.

Collection & dissemination	Knowledge attic & publisher
Content	Hybrid
Nature	Technical lessons
Orientation	Group of organizations and community
Duration	Permanent
Architecture	Standalone Integrated (Reactive and proactive dissemination)
Attributes & format	Textual and non-textual attributes
Confidentiality	Classified and restricted - Unclassified
Size	Huge

Table 15.1: Classification of our LL system with additional features.

15.1 Broker architecture

A problem with LL systems is that it might take time before significant profits become visible. Indeed, it usually lasts a long time before a LL repository contains a **critical and sufficient mass of knowledge**.

In the case of our prototype, it was impossible to reach this critical mass of knowledge in a such period of time (i.e. four months). Because of this constraint, it is very difficult to say whether our prototype is usable.

A possible solution could be based on interaction and knowledge exchange between different KM systems, as interaction with **comp.risks** or distribution and interaction throughout **brokers** and **distributed databases**.

On the other hand, a possible drawback could be the lack of confidence and the competitiveness between the various systems connected through the broker architecture.

15.1.1 Interaction with comp.risks forum

For example, in the case of interaction with a system like comp.risks forum, it should be interesting to gather and transfer new topics submitted in the forum to the LL system. And, the other way around, it could be interesting to gather and transfer new LL submitted in the LL system to the comp.risks forum. This type of process could be viewed and achieved as in the **active collection** (described in section 3.4). Therefore, extractions and transfers from comp.risks can be difficult, because of the non correspondence between the lack of structure of knowledge stored in comp.risks and the typology used in our LL system oriented towards safety-critical software.

This implies that our LL system is complementary with comp.risks but does not completely replace it. For example, it could be interesting to discuss a problem in comp.risks before to submit it in the LL system. It could allow to warn interested people earlier, while information is not yet complete and structured.

It is important to note that the current architecture of our LL system is not convenient for the additional purposes presented in this section. For example, in the case of distributed systems with brokers, technologies such as JAVA and CORBA would be more efficient, powerful and adapted.

15.1.2 Peer-to-peer knowledge management

Another possibility of interaction between different systems is the **Peer-to-peer knowledge management**[1], which is more able to fit with the emerging distributed organization of knowledge and lessons. It could be an interesting way to deploy distributed LL solutions.

15.2 Generic LL systems and specific format

15.2.1 Specific format

If interaction takes place among several LL systems, it is important for the LL exchanged to be understandable by all the different users. A user, who is used to work with such a LL system, should understand LL coming from other systems. Otherwise, he will be disappointed and reluctant to use the system.

This problem could probably be solved by defining a **specific format** to exchange data between different LL systems. The different organizations should therefore agree on this format and adapt their existing KM systems. It could be interesting to define a common **ontology** to exchange data between LL systems.

15.2.2 Generic LL systems

The LL system presented in this master thesis is suitable for software accidents related to the software community. However, the typology of these LL and related accidents (industrial sectors, consequences, life cycles, relevant sources, products/services, accident event sequence...) is currently fixed. It would be more useful if the prototype could be **adaptable** and **flexible** to any kind of organizations.

The LL system could be more **generic** and allow **tuning** to a **specific organization**, in such a way that a typical organization could define its own LL and accident typology. It could be a kind of **meta-model** or **evolvable ontology**. Once an organization installs the LL system, the system would ask to define the wished typology and the values of each attribute, as presented in figure 11.30 of section 11.7. Once the LL system installed, administrators could manage it as they want.

[1] http://www.p2pkm.org

Chapter 16

Positive as well as negative experiences

Several authors, such as [Weber & al., 2001] and [Silva & al., 2002], disagree with the fact of mixing different kinds of knowledge artifacts (described in subsection 3.2.2) in the same KM system. However, recent experiments [GAO, 2002] with LL systems show that mixing positive and negative experiences could promote reuse, or increase effectiveness and retrieval of relevant lessons.

LL can be built on **positive** or **negative experiences**. Nevertheless, if an organization concentrates only on failures, the general efficiency of its LL system could be decreased and it could miss opportunities to enhance all its processes, because users can be interested in positive as well as negative experiences. Sometimes, positive LL can even be more helpful than the negative ones, because users could try to imitate successes. NASA has noted it through his LL system [GAO, 2002].

Consequently, we think that, if we had to make the prototype again, we would include positive **(best practices)** as well as negative experiences. It is important not to focus only on accidents and problems. We also think that we would maybe remove the limitation that consists of allowing users to submit only lessons for which they have a solution **(incident reports)**. In case of urgency, it could also be very useful to submit **alerts**. These observations justify the type of our prototype which is throwaway and exploratory.

In subsection 3.2.2, we noted that alerts, best practices and incident reports are not considered as LL. In order to confirm or invalidate the aforementioned observations, a case study could be helpful and would certainly reveal others needs of extension or modification.

Chapter 17

Monitored distribution

In order to be more efficient, LL systems should be **incorporated** into the processes they intend to support [Weber & al., 2001]. In this chapter, advantages of integrating LL systems in decision support systems are discussed.

Although there seems to be benefits of using active (e.g. by mail with user profile subscription), reactive (e.g. by macro call in Word or Excel), or proactive (developed in this chapter) dissemination, these techniques have been implemented and tested in a small number of organizations [Weber & al., 2001]. It could be more useful to embed these LL systems and dissemination processes in the decision support systems targeted by their LL, as suggested in figure 17.1.

17.1 Features of monitored distribution

Active dissemination with monitored distribution is called **proactive dissemination**. This new concept consists of providing LL only *"when and where"* they are needed. In the monitoring, **distribution is strongly integrated with the targeted organizational processes/applications**. It is hoped that monitored distribution could improve the quality of the latter.

In order to give a simple example, the **Microsoft Office Companion** could be viewed as a sort of monitored distribution.

[Weber & al., 2002] claims that monitored distribution yields the following advantages:

- Distribution is tightly integrated with the targeted organizational processes.

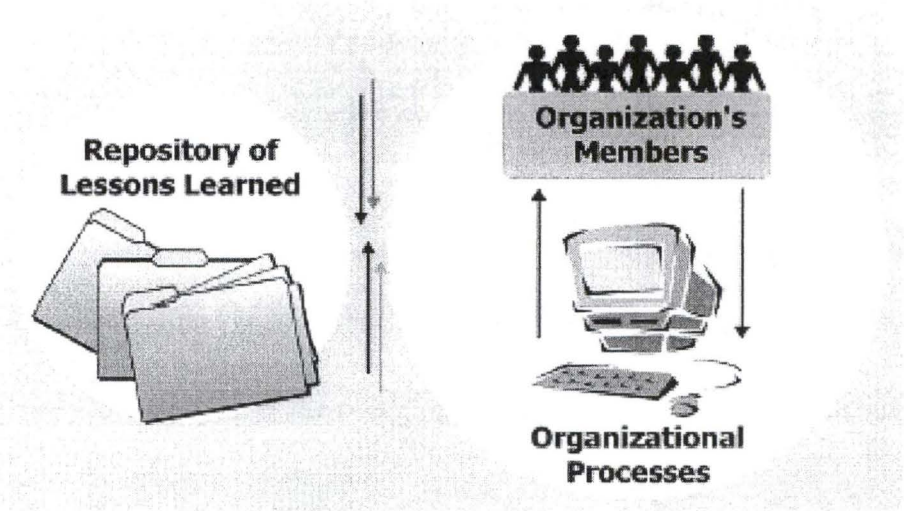

Figure 17.1: The lesson distribution gap [Weber & al., 2002].

- Users need not know or be reminded of the repository to use lessons, nor require lesson retrieval skills.
- Users do not need significant additional time to retrieve lessons.
- Because lessons are integrated with the targeted processes, interfaces can be developed with the monitored distribution approach to allow users to execute lesson suggestions.

In other words, LL systems with monitored distribution must play the following roles to achieve these goals:

- Identifying the best moment to deliver lessons.
- Anticipating the user needs.
- Providing an API to embed monitored distribution in another system.
- Asking the user for the state of unknown variables. This could happen, for example, if there is no sufficient conditions to justify lesson applicability. It allows to assess the similarity between the current conditions and a potentially applicable lesson.

Monitored distribution can use **AI techniques** like text indexing, such as **latent semantic analysis**[1] (theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text). But AI techniques also have some

[1] http://LSA.colorado.edu/

limitations. For example, understanding and correctly interpreting natural language still remain challenging problems. Moreover, if the matching is random and/or too much intrusive, such as the Microsoft Office Companion for example, there is a risk for the tool to get deactivated by the user.

17.2 Example of architecture for embedded LL systems

[Weber & al., 2002] gives an example of monitored distribution that consists of integrating a LL system, the Active Lesson Delivery System (ALDS), as a module of a Decision Support System (DSS). The requirements for the integration are that the DSS has a flexible architecture and that the decision/task/process and state conditions that determine decision making are explicitly represented (i.e., in such a way that an applicability oriented retrieval process can be used to distribute lessons).

Figure 17.2 shows the interaction between ALDS and DSS. The inputs of the DSS concern what the user currently performs. ALDS keeps track of the state conditions input by the user and uses them plus the current task to assess and compare it with LL stored in the repository. If a lesson is considered to be sufficiently similar to the current situation and applies to the current task, then ALDS considers it to be applicable. It displays it to the user, in such a way that he can take decisions from it.

However, DSS is particularly used in company management (US Army for example). In the context of software engineering, case tools are more useful. **PRIME** (PRocess-Integrated Modeling Environments) is a process-integrated environment (PIE) [Pohl & al., 1999]. It consists of a workflow system in which we can explicitly represent and execute the process. The piece of process executed at a given time depends on the formalized context. The process-integrated tools of PRIME adjust their behavior according to the current process situation and the method definitions. In the case of a LL system, a system like PRIME could allow to match the context with LL applicability and the associated process with the solution of the LL.

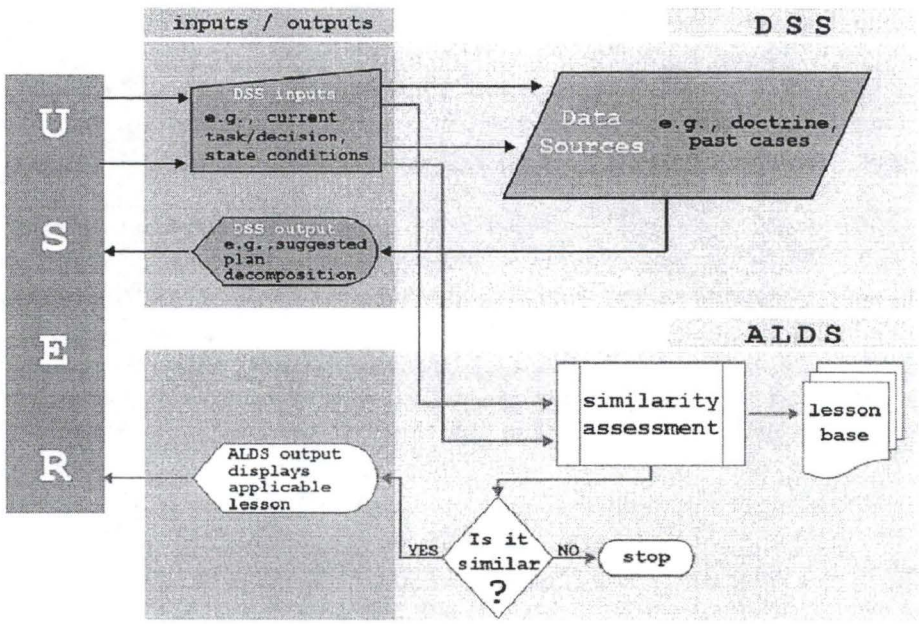

Figure 17.2: An architecture for integrating monitored distribution in a decision support system [Weber & al., 2002].

Chapter 18

Other perspectives of research

18.1 LL accessibility rights

In some cases, it could be interesting to define **confidentiality** with some **accessibility rights** on LL. For example, a specific LL could be viewed by all users, or only by a specific group of users (air force technicians, or nuclear engineers for example). It could also be only edited by a specific user, the submitter, or by a group of users/experts. Such right management will raise confidence in the system.

18.2 Confrontation of experts' opinions

Concerning the level of validating a LL, it could be interesting to implement a system that allows different experts to discuss a specific LL and exchange their opinion and advice. In the future, we can consider a system which will allow many experts to validate a LL and his solution, in the aim of strengthen it. The discussion between experts could be completed with a vote system. Experts would be authorized to vote for such or such opinions. It could allow to reduce conflicts between experts.

In order to designate experts of the LL system, it could also be useful to imagine a vote system between existing experts.

Conclusion

In this master thesis, which comes within the scope of the works of [Silva & al., 2002], we discussed the requirements of LL systems and built an exploratory throwaway prototype to collect, validate, and disseminate lessons related to accidents of safety-critical software products and devices. This prototype allowed us to explore and understand the functioning of such LL systems designed to be used by software engineers.

Because of the non existence of LL systems oriented towards safety-critical software, we decided to opt for a new and innovative approach based on KM.

After having given some examples and discussed current problems of software accident reporting, we illustrated how the use of KM brings innovation and allows improvements in the design of LL systems. We continued by explaining the problems the setting up and the management of a LL system can face, and by giving some means and ideas to improve and guarantee its success. Finally, we dealt with ethical questions concerning LL systems.

In the previous part about possible future works, we threw interesting ideas, which should be taken into account, in order to improve the features and the efficiency of LL systems. It concerns perspectives of research that could have been considered if we had more time to develop a more complete and powerful LL system.

We hope that this reading was interesting for you and that you have learned some lesson ;-)

We thank you for the interest you have expressed by reading this master thesis.

Bibliography

[Bartlett, 1999] Bartlett, J. (1999). Unpublished slides.

[Bickford, 2000] Bickford, J. (2000). Sharing lessons learned in the Department of Energy. In Aha, D. W., Weber, R., *Intelligent lessons learned systems: Papers from the AAAI Workshop (Technical Report WS-00-03)*, pp. 5-8. AAAI Press, Menlo Park, Canada.

[Bowie & al., 1990] Bowie, N., Duska, R. (1990). *Business Ethics*, pp. 10-11. Prentice Hall, Englewood Cliffs, New Jersey, USA.

[Cockburn, 2000] Cockburn, A. (2000). *Writing Effective Use Cases.* Addison-Wesley, Boston, USA.

[DOE-STD-7501-99, 1999] DOE STANDARD (1999). *The DOE Corporate lessons learned program.* Department of Energy, Washington, DC. Available at http://www.tis.eh.doe.gov/ll/docs.html (Date of access 02/05/2004).

[Dumas & al., 2003] Dumas, P., Bulinge, F., Boutin, E. (2003). *Ethical dimebsions of KM in professional settings.* Laboratoire LePont, Université de Toulon-Var, France.

[Fisher & al., 1998] Fisher, D., Deshpande, S., & Livingston, J. (1998). *Modeling the lessons learned process* (Research Report 123-11). Albuquerque, University of New Mexico, Department of Civil Engineering.

[GAO, 2002] United States General Accounting Office (2002). Report to the Subcommittee on Space and Aeronautics, Committee on Science, House of Representatives. *NASA: Better Mechanisms Needed for Sharing Lessons Learned.* Available at http://www.gao.gov/new.items/d02195.pdf (Date of access 02/05/2004).

[Jackson, 2001] Jackson, M. (2001). *Problem Frames: Analyzing and structuring software development problems.* Addison-Wesley/ACM Press, New York, USA.

[Johnson & al., 2000] Johnson, C., Birnbaum, L., Bareiss, R., & Hinrichs, T. (2000). War stories: harnessing organizational memories to support task performance. *Intelligence: New Visions of AI in Practice*, Volume 11 , Issue 1, pp. 17-31. Addison-Wesley/ACM Press, New York, USA.

[Knight & al., 2001] Knight, J., Leveson, N., DeWalt, M., Elliot, L., Kaner, C., & Nissembaum, H. (2001). *On Licensing Software Engineers Working on Safety-Critical Software.* Association for Computing Machinery. Available at http://www.acm.org/serving/se_policy/safety_critical.pdf (Date of access 02/05/2004).

[Knight & al., 2000] Knight, C., & Aha, D. W. (2000). A common knowledge framework and lessons learned module. In Aha, D. W., & Weber, R., *Intelligent lessons learned systems: Papers from the AAAI Workshop (Technical Report WS-00-03)*, pp. 25-28. AAAI Press, Menlo Park, Canada.

[Leveson, 1995] Leveson, N. (1995). *Safeware: System Safety and Computers.* Addison-Wesley/ACM Press, New York, USA.

[Minsky, 1996] Minsky, M. (1996). Intelligent machines. In Brockman, J., *The Third Culture: Beyond the Scientific Revolution.* Touchstone Books.

[Neumann, 1995] Neumann, P. G. (1995). *Computer Related Risks.* Addison-Wesley/ACM Press, New York, USA.

[Peterson, 1996] Peterson, I. (1996). *Fatal Defect: Chasing Killer Computer Bugs.* Vintage Books, New York, USA.

[Petroski, 1994] Petroski, H. (1994). *Design Paradigms: Case Histories of Error and Judgment in Engineering.* Cambridge University Press, Cambridge, UK.

[Pohl & al., 1999] Pohl, K., Weidenhaupt, K., Dömges, R., Haumer, P., Jark, M., Klamma, R. (1999). PRIME: Towards Process-Integrated Environments. *ACM Transaction on Software Engineering and Methodology*, Volume 8, Issue 4. Lehrstuhl Informatik V (Information Systems), Aachen, Germany.

[Powers, 1999] Powers, V. J. (1999). Xerox creates a knowledge-sharing culture through grassroots efforts. *Knowledge management - In practice*, Issue 18. American Productivity and Quality Center, Houston, USA. Available at http://www.askmecorp.com/pdf/Xerox.pdf (Date of access 02/05/2004).

[Reimer, 1998] Reimer, U. (1998). *Knowledge integration for building organizational memories.* Swiss Life, Information Systems Research Group. Available at http://www.dfki.uni-kl.de/ aabecker/Freiburg/Final/Reimer/ki97-ws/ki97-ws.html (Date of access 02/05/2004).

[Rus & al., 2002] Rus, I., & Lindval, M. (2002). Knowledge Management in Software Engineering. *IEEE Software*, Volume 19, Issue 3, pp. 26-38. Available at http://fc-md.umd.edu/mikli/RusLindvallKMSE.pdf (Date of access 02/05/2004).

[Secchi & al., 1999] Secchi, P., Ciaschi, R., & Spence, D. (1999). A Concept for an ESA lessons learned system. In Secchi, P., *Proceedings of Alerts and LL: An Effective Way to Prevent Failures and Problems (Technical Report WPP-167)*, pp. 57-61. ESTEC, Noordwijk, The Netherlands.

[Senge, 1994] Senge, P. M. (1994). *The Fifth Discipline: The Art and Practice of the Learning Organization*, pp. 139. Paperback, Currency/Doubleday, New York, USA.

[Silva & al., 2002] Silva, A., Maté, J. L., & Pazos, J. (2002). *Lessons learned systems for critical software: Introduction and perspectives.* Unpublished article.

[Silva, 2003] Silva, A. (2003). *Sistemas Software Críticos.* Unpublished lesson.

[Stewart, 1997] Stewart, T. A. (1997). *Intellectual capital: the new wealth of organizations.* Paperback, Currency/Doubleday, New York, USA.

[Tah & al., 2001] Tah, J. H. M., & Carr, V. (2001). Towards a framework for project risk knowledge management in the construction supply chain. *Advances in Engineering Software*, Volume 32, Issue 10-11, pp. 835-846. Elsevier Science Ltd, Oxford, UK.

[Terry, 1991] Terry, G. J. (1991). *Engineering System Safety.* Mechanical Engineering Publications Ltd., London, UK.

[van Heijst & al., 1997] van Heijst, G., van der Spek, R., & Kruizinga, E. (1997). Corporate memories as a tool for knowledge management. *Expert Systems with Applications*, Volume 13, Issue 1, pp. 41-54. Elsevier Science Ltd, Oxford, UK.

[Weber & al., 2001] Weber, R., Aha, D. W., & Becerra-Fernandez, I. (2001). Intelligent lessons learned systems. *Expert Systems with Applications*, pp. 17-34. Elsevier Science Ltd, Oxford, UK.

[Weber & al., 2002] Weber, R., & Aha, D. W. (2002). Intelligent delivery of military lessons learned. *Decision Support System*, pp. 287-304. Elsevier Science Ltd, Oxford, UK. Available at http://www.aic.nrl.navy.mil/hicap/pubs-ills.html (Date of access 02/05/2004).

[Wenger & al., 2000] Wenger, E., & Snyder, W. M. (2000). Communities of Practice: The Organizational Frontier. *Harvard Business Review*, pp. 139-145. Harvard, USA. Available at http://www.lahvista.cz/ceconsortium.com/pdf/commprac.pdf (Date of access 02/05/2004).

[Wiener, 1993] Wiener, L. R. (1993). *Digital Woes: Why we Should not Depend on Software.* Addison-Wesley/ACM Press, New York, USA.